Jena Research Papers in Business and Economics

ABSALOM: Balancing assembly lines with assignment restrictions

Armin Scholl, Malte Fliedner, Nils Boysen

02/2008

Jenaer Schriften zur Wirtschaftswissenschaft

Working and Discussion Paper Series
School of Economics and Business Administration
Friedrich-Schiller-University Jena

ISSN 1864-3108

Publisher:

Wirtschaftswissenschaftliche Fakultät
Friedrich-Schiller-Universität Jena
Carl-Zeiß-Str. 3, D-07743 Jena

www.jbe.uni-jena.de

Editor:

Prof. Dr. Hans-Walter Lorenz
h.w.lorenz@wiwi.uni-jena.de

Prof. Dr. Armin Scholl
armin.scholl@wiwi.uni-jena.de

www.jbe.uni-jena.de

ABSALOM: Balancing assembly lines with assignment restrictions

Armin Scholl

Chair of Decision Analysis and Management Science, Friedrich-Schiller-University Jena, Carl-Zeiß-Straße 3, D-07743 Jena, e-Mail: armin.scholl@wiwi.uni-jena.de (corresponding author)

Malte Fliedner

Institute of Industrial Management, University of Hamburg, Von-Melle-Park 5, D-20146 Hamburg, e-Mail: fliedner@econ.uni-hamburg.de

Nils Boysen

Chair of Operations Management, Friedrich-Schiller-University Jena, Carl-Zeiß-Straße 3, D-07743 Jena, e-Mail: nils.boysen@uni-jena.de

Assembly line balancing problems (ALBPs) arise whenever an assembly line is configured, redesigned or adjusted. An ALBP consists of distributing the total workload for manufacturing products among the work stations along the line. On the one hand, research has focussed on developing effective and fast solution methods for exactly solving the simple assembly line balancing problem (SALBP). On the other hand, a number of real-world extensions of SALBP have been introduced but solved with straightforward and simple heuristics in many cases. Therefore, there is a lack of procedures for exactly solving such generalized ALBP.

In this paper, we show how to extend the well-known solution procedure SALOME *(INFORMS J. Computing 9, 319-334)*, which is able to solve even large SALBP instances in a very effective manner, to a problem extension with different types of assignment restrictions (called AR-ALBP). The extended procedure is given the acronym ABSALOM. It consists of a favourable branching scheme, an arsenal of bounding rules and a variety of logical tests using ideas from constraint programming.

Computational experiments show that ABSALOM is a very promising exact solution approach though the additional assignment restrictions complicate the problem and require some components of SALOME to be modified in a relaxing manner.

Keywords: assembly line balancing; assignment restrictions; combinatorial optimization; branch-and-bound

1. Introduction and notation

Assembly lines are flow-oriented production systems which are typical in the industrial production of high quantity standardized commodities and even gain importance in low volume production of customized products. Among the decision problems which arise in managing such systems, assembly line balancing problems are important tasks in medium-term production planning (cf., e.g., Baybars 1986; Becker and Scholl 2006; Boysen et al. 2007).

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

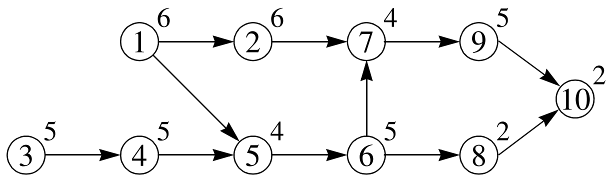
Figure 1. Precedence graph

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

The following sets are useful to describe the precedence relations:

- $P_i := \{h \mid (h,i) \in E\}$ set of direct predecessors of task $i \in V$
- $F_i := \{j \mid (i,j) \in E\}$ set of direct successors (followers) of task $i \in V$

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

- $P_i^* := \{h \mid (h,i) \in E^*\}$ set of all predecessors of task $i \in V$
- $F_i^* := \{j \mid (i,j) \in E^*\}$ set of all successors of task $i \in V$

Any type of ALBP consists in finding a feasible *line balance*, i.e., an assignment of each task to a station such that the precedence constraints and possible further restrictions are fulfilled. The set S_k of tasks assigned to a station k (=1,...,m) constitutes its *station load*, the cumulated task time $t(S_k) := \sum_{j \in S_k} t_j$ is called *station time*. When a fixed common cycle time c is given, a line balance is feasible only if the station time of neither station exceeds c. In case of $t(S_k) < c$, the station k has an *idle time* of $c - t(S_k)$ time units in each cycle, i.e., it is repeatedly unproductive for this time span.

The most popular ALBP is called *Simple Assembly Line Balancing Problem (SALBP)*. It has the following characteristics (cf. Baybars 1986; Scholl 1999, ch. 2.2; Boysen et al. 2007):

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

Several problem versions arise from varying the objective as shown in Table 1. The tuple-notations specify the characterizations of the problem versions within the recent classification scheme of Boysen et al. (2007).

Since SALBP-F is an NP-complete feasibility problem, the optimization versions of SALBP, which may be solved by iteratively examining several instances of SALBP-F, are NP-hard (cf. Wee and Magazine 1982; Scholl 1999, ch. 2.2.1.5). Recent surveys covering SALBP models and procedures are given by Erel and Sarin (1998), Scholl (1999, ch. 2, 4, 5), Rekiek et al. (2002b) as well as Scholl and Becker (2006).

Table 1. Versions of SALBP

		cycle time c	
		given	minimize
no. m of stations	given	SALBP-F $[\ \vert\ \vert\ ]$	SALBP-2 $[\ \vert\ \vert c]$
	minimize	SALBP-1 $[\ \vert\ \vert m]$	SALBP-E $[\ \vert\ \vert E]$

In this paper, we focus on SALBP-1 and its extension to assignment restrictions defining the new problem ARALBP-1, but the same transformations apply to the other versions of the problem. The remainder of the paper is organized as follows: In Section 2, different types of assignment restrictions are described and surveyed leading to the extended line balancing problem. A mathematical program for this problem is given in Section 3. Section 4 is devoted to describing the exact solution procedure ABSALOM that is based on the well-established procedure SALOME for SALBP-1. Computational experiments are reported in Section 5. A summary and statements on future research issues in Section 6 conclude the paper.

2. Assignment restrictions

In practice, there are usually constraints which restrict the assignment of tasks to stations in addition to the cycle time constraint and the precedence relations. The following types of assignment restrictions are considered in literature (cf. Boysen et al. 2007; cf. Scholl 1999, ch. 1.3.4):

- **Task restrictions**, also called *zoning constraints,* either *link* a set of tasks which have to be assigned to the same station ($\alpha_5 = \text{link}$ in the classification scheme of Boysen et al. 2007) or ensure that *incompatible tasks* are assigned to different stations ($\alpha_5 = \text{inc}$). Especially, if resources are not explicitly considered, task restrictions are a convenient way to make sure that tasks which require an expensive resource share the same station (e.g. Dar-El and Rubinovitch 1979) or that tasks which require different equipment that cannot be provided at the same station are assigned to different stations.

- **Resource (attribute) restrictions:** Furthermore, the assignment of tasks to a station might be subject to constraints on the *cumulated* value of particular task attributes ($\alpha_5 = \text{cum}$). For example, the space for placing machines or material boxes as well as for storing material or tools, which are required for performing the tasks, might be scarce. Then, station loads are only acceptable if the cumulated space needs of required resources is not exceeding the available space (e.g., Kim and Park 1995; Pastor et al. 2002; Sawik 2002; Bautista and Pereira 2007). Another constraint might concern the cumulated requirement on the operators' grip strengths which must not exceed a certain threshold from an ergonomic point of view (Carnahan et al. 2001).
- **Station restrictions:** Regularly, there are restrictions on assigning tasks to certain stations. Such restrictions are particularly relevant when the line is already installed and should be rebalanced without rearranging all the equipment. For example, tasks can be restricted to a certain type of stations ($\alpha_5 = \text{type}$; Johnson 1983). This might be the case, if workpieces need to undergo position changes while they are processed. Then, a task can only be assigned to a station where the workpiece is in the position required (Lapierre and Ruiz 2004). Tasks which need heavy machinery or handling equipment fixed to a certain station must be performed there ($\alpha_5 = \text{fix}$). In other cases, stations might have conditions that prevent a task from being carried out ($\alpha_5 = \text{excl}$).
  In general, the assignment of a task might be restricted to a certain zone of the line (station interval) or to a set of single stations that provide the conditions required.
  Remark: While Johnson (1983) considers the type of a station as a variable (for each station a type is to be chosen), $\alpha_5 = \text{type}$ assumes that for each station the type is already predetermined such that tasks which require another type are excluded from the respective stations. As a consequence, $\alpha_5 = \text{type}$ (and also $\alpha_5 = \text{fix}$ as a special case with each station being another type) can be modelled as $\alpha_5 = \text{excl}$ by simply excluding all stations that do not fulfill the conditions. The Johnson approach can be modelled using $\alpha_5 = \text{inc}$ such that tasks which require different station types are incompatible with each other.
- **Distance restrictions:** Sometimes, the production process can also require to observe *minimum distances* ($\alpha_5 = \min$) or *maximum distances* ($\alpha_5 = \max$) between tasks measured in time, space, sequence or station positions (see Buxey 1974; Pastor and Corominas 2000). A minimum distance, e.g., has to be observed in cases where color or glue has to dry before further tasks can be performed at the corresponding area on the workpiece. A maximum distance is, e.g., be defined when melted metal must be prevented from cooling down before a certain other task is carried out.

Table 2 gives an overview on the research papers dealing with assignment restrictions. Obviously, the task restrictions are seen to be most relevant, while distance restrictions have not attracted considerable attention, because there seem to be only very few realistic examples for minimum or maximum distances covering more than two neighbouring stations. Thus in most cases, the simpler task restrictions (linking which expresses a maximum distance of 0 and incompatibility which expresses a minimum distance of one station) are sufficient special cases. This lack of importance is accompanied with a considerable effort for including general distance restrictions into a solution procedure as will be discussed later.

Additionally, Table 2 summarizes the results of a poll among 34 enterprises of the automobile, electronics and machine construction industries which run one or more assembly lines. The last column of the table states the numbers of those firms where the respective assignment restrictions occur. Obviously, all restriction types are really existant and most are of significant importance. Though minimum distances are mentioned by 13 firms and maximum distances, by 4 firms, the application details make clear that in almost every case a minimum distance of type "inc" and a maximum distance of type "link" is meant.

Table 2. Survey of research on assignment restrictions

α_5	references	firms
link	Bhattacharjee and Sahu (1987), Boysen and Fliedner (2008), Buxey (1974), Deckro (1989), Gökcen and Erel (1997), Hautsch et al. (1972), Klenke (1977), Lapierre and Ruiz (2004), Miralles (2005), Pastor and Corominas (2000), Pinnoi and Wilhelm (1997), Rekiek et al. (2001, 2002), Roberts and Villa (1970), Schofield (1979), Vilarinho and Simaria (2002, 2006), Zäpfel (1975)	13
inc	Agnetis et al. (1995), Bautista and Pereira (2002), Bautista et al. (2000), Bhattacharjee and Sahu (1987), Boysen and Fliedner (2008), Buxey (1974), Deckro (1989), Gökcen and Erel (1997), Klenke (1977), Lapierre and Ruiz (2004), Park et al. (1997), Pastor and Corominas (2000), Pinnoi and Wilhelm (1997), Rekiek et al. (2001), Schofield (1979), Vilarinho and Simaria (2002, 2006), Zäpfel (1975)	7
cum	Arcus (1966), Bautista and Pereira (2007), Boysen and Fliedner (2008), Carnahan et al. (2001), Kim and Park (1995), Liu and Chen (2002), Miralles (2005) Pastor et al. (2002), Pinnoi and Wilhelm (1997), Sawik (2002), Wilhelm and Gadidov (2004)	18
fix	Agnetis et al. (1995), Arcus (1966), Bartholdi (1993), Bhattacharjee and Sahu (1987), Kim et al. (2000), Leu et al. (1994), Malakooti and Kumar (1996), Miltenburg (1998), Pastor et al. (2002), Raouf and Tsui (1982), Rekiek et al. (2001, 2002), Schofield (1979), Vilarinho and Simaria (2006)	19
excl	Raouf and Tsui (1982), Schofield (1979)	6
type	Bartholdi (1993), Gadidov and Wilhelm (2000), Johnson (1983, 1991), Kim et al. (2000), Lapierre and Ruiz (2004), Lapierre et al. (2006), Lee et al. (2001), Pastor and Corominas (2000), Pinnoi and Wilhelm (1997), Rekiek et al. (2001)	27
min	–	13
max	Buxey (1974), Deckro (1989), Pastor and Corominas (2000)	4

Up to now, a lot of heuristic procedures and a few exact approaches have been proposed for solving ALBP with a selection of the assignment restriction mentioned above. But there is no exact solution procedure available for the case of simultaneously considering several types of assignment restrictions.

Extending the simple assembly line balancing problem of type 1 (SALBP-1) by adding the different types of assignment restrictions, we get a problem which we call **ARALBP-1** (**A**ssignment **R**estricted **A**ssembly **L**ine **B**alancing **P**roblem of type **1**). In the recent classification scheme of Boysen et al. (2007), ARALBP-1 is denoted as **[link, inc, cum, excl, fix, type, min, max| | m]**. Even if only a subset of the restriction types are present, we will use the same name of the problem. The other problem versions (cf. Table 1) are to be defined accordingly.

3. Mathematical program for ARALBP-1

In the following, we extend the standard mathematical program for SALBP-1 (cf. Bowman 1960, White 1961, Thangavelu and Shetty 1971, Patterson and Albracht 1975, Scholl 1999, p. 29) by adding the different types of assignment restrictions to represent the extended problem ARALBP-1.

Other models which consider some of the assignment restrictions (mostly linked tasks or fixed stations, which are included easily) are given by, among others, Bautista and Pereira (2007), Deckro (1989), Gökcen and Erel (1997), Pastor and Corominas (2000), Pastor et al. (2004), Pinnoi and Wilhelm (1997), Vilarinho and Simaria (2006).

The proposed integer linear program is based on *binary assignment variables* x_{jk} and a valid upper bound $\overline{m}$ on the number of stations (cf. Scholl 1999, ch. 2.2.2.2):

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

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

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

$$E_j := \left\lceil \tau_j + \sum_{h \in P_j^*} \tau_h \right\rceil \quad \text{and} \quad L_j(\overline{m}) := \overline{m} + 1 - \left\lceil \tau_j + \sum_{h \in F_j^*} \tau_h \right\rceil \qquad \text{for } j \in V \tag{1}$$

Thus, task j can only be assigned to one of the stations out of the set of feasible stations $FS_j = \{E_j, E_j + 1, \ldots, L_j(\overline{m})\}$. The station sets can be used to reduce the number of variables such that x_{jk} has only to be defined for $j \in V$ and $k \in FS_j$. As a consequence, only a subset $B_k = \{j \in V \mid k \in FS_j\}$ of the tasks are *potentially assignable* to the stations $k = 1,\ldots,\overline{m}$.

To observe the **station restrictions**, the station sets FS_j have to be reduced by deleting all stations to which an assignment of task j is not allowed ($\alpha_5 \in \{\text{excl}, \text{type}\}$). In case that only a single station is allowed ($\alpha_5 = \text{fix}$), the set contains only this specific station. If the resulting set is empty, the problem has no feasible solution due to contradicting precedence relations and station constraints.

A **minimum distance** $\underline{d}_{ij} \geq 2$ between two tasks i and j requires that the stations k_i and k_j, to which i and j are assigned, fulfill $|k_j - k_i| \geq \underline{d}_{ij}$. Similarly, a **maximum distance** $\overline{d}_{ij} \geq 1$ between tasks i and j requires that the inequality $|k_j - k_i| \leq \overline{d}_{ij}$ holds. If in either case the relation $k_i < k_j$ is additionally required, this is achieved by imposing the precedence relation (i, j).

Notice that $\underline{d}_{ij} = 1$ would only state that i and j are *incompatible*. Similarly, $\overline{d}_{ij} = 0$ represents *linked* tasks. In order to define consistent distance requirements, $\underline{d}_{ij} \leq \overline{d}_{ij}$ must hold for all task pairs. As non-restricting default values, we might use $\underline{d}_{ij} = 0$ and $\overline{d}_{ij} = \overline{m} - 1$.

Due to the topological numeration of vertices and the symmetry of the distance measurement, it is sufficient to consider task relationships with $i<j$. In the following, this is always assumed unless the opposite is stated explicitly. The following sets of task pairs are distinguished:

IT	set of all pairs of tasks $i,j \in V$ which are *incompatible* to each other
LT	set of all pairs of tasks $i,j \in V$ which are *linked* to each other
LD	set of all pairs of tasks $i,j \in V$ with *minimum (lower bound) distance* $\underline{d}_{ij} \geq 2$
UD	set of all pairs of tasks $i,j \in V$ with *maximum (upper bound) distance* $\overline{d}_{ij} \geq 1$

Following parameters are used to describe the **resource restrictions**:

R	set of scarce resources (e.g., storage space, space for machinery, manual labor)
A_r	capacity of resource $r \in R$ available at each station (e.g., storage space available, number of tool places at a robot, physical capacity of manual labor per cycle)
u_{jr}	usage of resource r by task $j \in V$ (e.g., storage space required for the parts mounted by task j, number of tools required by task j, amount of physical labor capacity necessary for performing task j); relative resource usage $\tau_{jr} = u_{jr} / A_r$

Since the time can also be seen as a scarce resource, we may interpret it as a resource 0 with $A_0 = c$, $u_{j0} = t_j$ and $\tau_{j0} = \tau_j$. The set of resources is extended to $R_0 = R \cup \{0\}$. Using this transformation, we can compute strengthened earliest and latest stations as well as more restricted sets FS_j as follows:

$$E_j := \max\left\{ \left\lceil \tau_{jr} + \sum\nolimits_{h \in P_j^*} \tau_{hr} \right\rceil \middle| r \in R_0 \right\} \qquad \text{for } j \in V \qquad (2)$$

$$L_j(\overline{m}) := \overline{m} + 1 - \max\left\{ \left\lceil \tau_{jr} + \sum\nolimits_{h \in F_j^*} \tau_{hr} \right\rceil \middle| r \in R_0 \right\} \qquad \text{for } j \in V \qquad (3)$$

Binary linear program for ARALBP-1

The **objective function** minimizes the index m of the station the sink node n is assigned to.

$$\text{Minimize } m(\mathbf{x}) = \sum_{k \in FS_n} k \cdot x_{nk} \qquad (4)$$

such that the following restrictions are fulfilled.

Occurrence and station restrictions: Each task is assigned to exactly one station out of the feasible station set FS_j.

$$\sum_{k \in FS_j} x_{jk} = 1 \qquad \text{for all } j \in V \qquad (5)$$

Cycle time restrictions: The station time (sum of operation times of tasks assigned) must not exceed the cycle time in any station k.

$$\sum_{j \in B_k} t_j \cdot x_{jk} \leq c \qquad \text{for } k = 1,...,\overline{m} \qquad (6)$$

Resource restrictions: Concerning each resource r, the joint (additive) requirement of tasks assigned to station k must not exceed the capacity of that resource available at k. Obviously, these restrictions are of the same type as the cycle time constraint such that (6) could be included in (7) by choosing $r \in R_0$.

$$\sum_{j \in B_k} u_{jr} \cdot x_{jk} \leq A_r \qquad \text{for } k = 1,...,\overline{m} \text{ and } r \in R \qquad (7)$$

Precedence relations: A task j must not be assigned to an earlier station than its predecessor i. This needs only be checked if the station intervals overlap.

$$\sum_{k \in FS_i} k \cdot x_{ik} \leq \sum_{k \in FS_j} k \cdot x_{jk} \qquad \text{for all } (i,j) \in E \text{ with } L_i > E_j \qquad (8)$$

Incompatible tasks i and j must not be assigned to the same station k in case that this is actually possible.

$$x_{ik} + x_{jk} \leq 1 \qquad \text{for all } (i,j) \in IT \text{ and } k \in FS_i \cap FS_j \qquad (9)$$

Linked tasks i and j must be assigned to the same station. Obviously, if the tasks have no possible station in common, i.e., $FS_i \cap FS_j = \varnothing$, no feasible solution exists.

$$\sum_{k \in FS_i} k \cdot x_{ik} = \sum_{k \in FS_j} k \cdot x_{jk} \qquad \text{for all } (i,j) \in LT \qquad (10)$$

Minimum distances: Precedence-related tasks i and j must be assigned to stations which differ at least by the minimum distance.

$$\sum_{k \in FS_j} k \cdot x_{jk} - \sum_{k \in FS_i} k \cdot x_{ik} \geq \underline{d}_{ij} \qquad \text{for all } (i,j) \in LD \cap E^* \qquad (11)$$

If the tasks are not related by precedence, all relevant station intervals of length $\underline{d}_{ij}$ are considered. Let T_{kij} be the station interval starting with station k, i.e., $T_{kij} = \{k, \ldots, k + \underline{d}_{ij} - 1\}$. Furthermore, let K_{ij} be the set of all relevant starting stations k where a station interval to be examined might start, i.e., $K_{ij} = \{\min\{E_i, E_j\}, \ldots, \max\{L_i(\overline{m}), L_j(\overline{m})\} - \underline{d}_{ij} + 1\}$. Using these sets, the minimum distance restrictions of unrelated tasks are formulated as:

$$\sum_{h \in FS_i \cap T_{kij}} x_{ih} + \sum_{h \in FS_j \cap T_{kij}} x_{jh} \leq 1 \qquad \text{for all } (i,j) \in LD - E^* \text{ and } k \in K_{ij} \qquad (12)$$

Note that by modelling the constraints in this manner, it is avoided to introduce additional binary variables to represent which task is performed earlier. Furthermore, note that the definition of set K_{ij} could be refined to reduce the number of constraints but this would result in an unnecessarily complicated presentation. The formulation given can be used for precedence-related tasks in a simplified manner but the restrictions (11) are more intuitive.

Maximum distances: Two tasks i and j with a maximum station distance must be assigned to stations which do not differ by more than the allowed value. If the tasks are related by precedence, inequality (13) is sufficient. If they are not related by precedence, the maximal distance must be guaranteed in the other direction, too, as modelled in (14). Note that one of both restrictions for such a task pair is automatically fulfilled due to a negative value at the left-hand side while the other restriction observes the maximum distance (or both left-hand sides are zero if the tasks are in the same station).

$$\sum_{k \in FS_j} k \cdot x_{jk} - \sum_{k \in FS_i} k \cdot x_{ik} \le \bar{d}_{ij} \qquad \text{for all } (i,j) \in UD \tag{13}$$

$$\sum_{k \in FS_i} k \cdot x_{ik} - \sum_{k \in FS_j} k \cdot x_{jk} \le \bar{d}_{ij} \qquad \text{for all } (i,j) \in UD - E^* \tag{14}$$

Variable definition: The binary variables representing the assignment of task j to station k (value 1) or not (value 0) are defined as follows.

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

4. ABSALOM – An exact solution procedure

SALBP-1 and, thus, its generalization ARALBP-1 are NP-hard optimization problems. Therefore, having formulated a mathematical program as done in Section 3 is usually not sufficient to get optimal solutions in case of complex real-world instances. Even though the speed of standard optimization software like XPress-MP, CPLEX or LINDO has increased dramatically in the last years, they are often not able to find an optimal solution in an adequate span of time (cf. Scholl et al., 2006, who performed comprehensive experiments for another extension of SALBP-1). As a consequence, there is a need for spezialized solution procedures which exploit the problem structure.

For optimizing SALBP-1, a lot of specialized procedures are available. Surveys are given in Baybars (1986), Ghosh and Gagnon (1989), Scholl (1999, ch. 2.2.2 and 4.1) as well as Scholl and Becker (2006).

Computational experiments showed that the procedure SALOME of Scholl and Klein (1997), improved by Scholl and Klein (1999), is one of the best exact solution methods available for SALBP-1 (cf. Scholl and Klein 1997; Sprecher 1999; Scholl and Klein 1999; Miltenburg 2006). Therefore, we take SALOME as the basis of developing a branch-and-bound procedure for ARALBP-1. The new procedure is called **ABSALOM** (**SALOME** for **A**ssignment **B**ounded problems) and intensively uses bound arguments, dominance rules and reduction procedures which utilize and combine concepts from both, combinatorial optimization and constraint programming.

In the following, we describe all components and steps of the new procedure. The components already included in SALOME are only described very shortly companied with comprehensive descriptions of the modifications necessary. For detailed descriptions of SALOME see Scholl and Klein (1997) as well as Scholl (1999, ch. 4.1.4).

Due to the minor importance of maximum distance restrictions (cf. Section 2) and the fact that those restrictions require different successful components of SALOME to be deactivated or relaxed, we only describe a version of ABSALOM that considers task, resource and station restrictions as well as minimal distances. The latter are restricted to the case that only tasks which are (directly or indirectly) related by precedence are subject to such distances. This restriction is not very limiting because minimal distances usually arise from such relationships.

4.1. Preprocessing

At first, the **minimum distances** are removed from the problem by modifying the problem data. For each pair $(i,j) \in LD \cap E^*$, the following is to be done: Generate a chain of articificial tasks $h_1,...,h_H$ with $H = \underline{d}_{ij} - 1$ and place them in-between i and j by introducing the arcs (i,h_1) and (h_H,j). Set durations and attribute usages of all artificial tasks to zero. Furthermore, set all pairs (i,h_1), (h_H,j) and (h_i,h_{i+1}) for $i = 1,\dots,H-1$ of neighbouring tasks incompatible. After having solved the modified problem, the additional tasks can simply be deleted from the solution.

Remark: Alternatively, the minimum distances could be integrated into the enumeration scheme, the bounding procedures and logical tests directly requiring different modifications. Instead, we choose the simple transformation approach described above, because these distances are rather rare in practice as stated in Section 2. Even if they occur, only a few such distances with small distance values (often not more than two or three stations) will be present. Then, only some additional artificial tasks have to be inserted which do not complicate enumeration and bound computation considerably. Any such task being available in a station to be loaded can be prefixed immediately without influencing the usual load enumeration at all (cf. Section 4.4).

Second, the precedence graph is modified by merging **linked tasks** into mega nodes. This has the advantage of getting a reduced number of nodes and precedence relations. Furthermore, the merged tasks have larger task times being advantageous with respect to bounding (in particular, heads and tails get strengthened; Section 4.3) and the success of dominance and (domain) reduction rules (e.g., the Jackson dominance rule; Section 4.4).

The following steps have to be performed to build the modified precedence graph $G' = (V',E')$:

(1) Set $V' := V$ and $E' := E$ to initialize the graph.

(2) Remove an arbitrary linked pair (i,j) from the set LT and initialize a set $J := \{i,j\}$.

(3) Add all tasks to J which are linked to any task already in J. Remove the corresponding task pairs from LT. Repeat this step, until no further task can be added to J.

(4) Add all tasks to J which are members of a path connecting two tasks $i,j \in J$.

(5) Build a new mega node with index h by merging all tasks of J. This requires transfering all properties and relationships of the tasks contained in J to node h:

- The task time and resource usages are computed by summing up the individual values: $t_h := \sum_{j \in J} t_j$, $u_{hr} := \sum_{j \in J} u_{jr}$ for all $r \in R$

- The mega node h inherits all precedence relationships from its members. For each arc $(q,j) \in E'$ with $j \in J$ and $q \in V'-J$, an arc (q,h) is added to E' and for each $(j,q) \in E'$ with $j \in J$ and $q \in V'-J$, an arc (h,q) is added, if not already contained in E'.
- The set of feasible stations of mega node h consists of all stations which are feasible for all tasks in J, i.e., $FS_h := \bigcap_{j \in J} FS_j$.
- Incompatibilities between tasks j from J with tasks q outside J need to be transferred to node h by $IT := IT \cup \{(q,h) \mid q \in (V'-J)$ for which $\exists\, j \in J$: $(q,j) \in IT\}$. To simplify presentation, we assume that for each pair $(q,j) \in IT$, the set IT also contains the reverse pair (j,q). This redundancy is also assumed for the other sets, if useful, and removes the necessity to differentiate between situations where $j<q$ and such where $j>q$.

(6) Update the node set, i.e., $V' := V' \cup \{h\} - J$, and remove all pairs involving the tasks $j \in J$ from E' and IT.

(7) Repeat the steps (2) to (6) until LT is empty.

(8) To remove redundant arcs from E', finally compute the transitive closure of the remaining graph and update the predecessor and successor sets accordingly. Renumber the tasks in V' according to a topological odering.

Contradicting requirements of assignment constraints might make a merging to mega nodes impossible. This is, e.g., the case whenever the sum of task times exceeds the cycle time, two tasks in J are additionally defined to be incompatible or have a positive minimum distance or $FS_h = \{\ \}$ for any mega node h. If such a contradiction is detected, the whole procedure can be terminated, since no feasible solution exists.

In order to apply the Jackson dominance rule, **potential dominances** have to be computed and stored in a set PD containing pairs (i,j) of tasks with i potentially dominating j (for the definition of potential dominances see Section 4.4).

To compute a **global lower bound** LB, all bound arguments of SALOME (LB_1 to LB_7) are applied. The maximum of those values serves as the initial LB. How these bounds are modified to cope with assignment restrictions is explained in Section 4.3.

The initial **global upper bound** is set to $UB = \overline{m}+1$ to indicate that a feasible solution has not been found yet. If no maximal number of stations $\overline{m}$ is given by the problem data, $\overline{m} = n$ is used.

4.2. Branching

In the following, we summarize the **branching scheme** of SALOME and describe how to modify it in order to solve ARALBP-1. For details see Scholl and Klein (1997) and Scholl (1999, ch. 4.1.4).

The original problem and each resulting subproblem are branched by building complete loads for a station k to form an enumeration tree each level of which corresponds to one station (*station-oriented branching*). In the basic branching scheme, the stations are built in forward direction such that the levels $k=1,2,...$ of the tree correspond to the stations k. The nodes are generated following the *laser search* (depth-first-search) principle, i.e., in each node of the current branch, only one descending node is built and developed at a time. At each revisit of a node another load is built and followed to the next level.

When searching for a station load to be branched, it is examined whether or not this load is compatible to the current (local) lower bound of the node just being inspected. This lower bound is based on the most simple lower bound argument for SALBP-1, $LB1 = \lceil t_{sum} / c \rceil$, which is obviuosly also valid for ARALBP-1 (for further bound arguments see Section 4.3). The sum of all task times must be divided among the stations observing that no more than c time units are available at each station. If the quotient is not integral, the value can be rounded up to the next integer, because only complete stations are possible. Depending on the current value LB of the lower bound, there is idle time (balance delay time) $BD(LB) = LB \cdot c - t_{sum}$ which is unavoidable. That is, in the root node (level k=0) each load for the first station the idle time of which is no larger than BD(LB), i.e., $c - t(S_1) \leq BD(LB)$, is compatible with the initial lower bound LB. All loads fulfilling this condition are branched the first time they are generated, because they do not need to enlarge the number of stations required, i.e., the current value of LB. The remaining loads form a second class, because they require at least LB+1 stations to be installed. These loads are considered for branching not before all loads of the first class have been examined and the (local) lower bound has been increased by one station.

In any node at a level $k > 1$ with local lower bound LLB, a sequence of loads for the stations $h = 1, ..., k$ have already been built and might consume parts of the balance delay. The remaining idle time is computed as $RI_k := BD(LLB) - \sum_{h=1}^{k} (c - t(S_h))$. Through the remaining idle time RI_k, the possible loads of station k+1 can also be subdivided into two classes. The first class contains all station loads with an idle time of at most RI_k time units; the second one contains all other possible station loads.

This technique of (locally) presorting the loads in each node is called *local lower bound method* (LLBM). It has the advantages to consider the most promising branches first and to strengthen lower bounds as soon as possible and has been proven to be very successful in accelerating the search by Scholl and Klein (1997, 1999).

Nevertheless, the sequence in which tasks are combined to trial station loads is still important, because the remaining tasks of a reduced problem decide on the possibilities to form loads for later stations. On the one hand, stations at the beginning of the line may be loaded with a small amount of idle time by combining tasks with small task times first. On the other hand, remaining large tasks possibly not able to share a station with each other may lead to very bad loads in later stations. In order to find good feasible solutions in the first branches of the tree, it is promising to prefer tasks with large task times and resource requirements and/or such which make available many successors. This is achieved by using a **dynamic renumbering** of tasks, i.e., in each node of the enumeration tree the tasks potentially assignable to the current station are renumbered in a manner which is favourable for finding promising loads first. On principle, the renumbering scheme based on priority values p_j and q_j introduced by Scholl and Klein (1999) is used. This scheme assures that the topological ordering is preserved. In order to account for assignment restrictions, the renumbering scheme has been modified. The priority values are computed such that all resources (recall that the cycle time can be interpreted as a resource) are considered with the scarcest one (with respect to utilization rate) defining the priority value, respectively (for computing earliest stations E_j and latest stations $L_j(m)$ see Section 3 and 4.3):

$$p_j := \frac{\max\{\tau_{jr} \mid r \in R_0\}}{L_j(LLB) - E_j + 1} \quad \text{and} \quad q_j := \max\left\{\tau_{jr} + \sum\nolimits_{i \in F_j^*} \tau_{ir} \mid r \in R_0\right\} \tag{16}$$

As recommended in Scholl and Klein (1999), the order strength of the precedence graph $OS = |E^*| / (n \cdot (n-1))$ is used to decide on which priority values are used for renumbering. If $OS < 0.4$, the priority values p_j (maximal average relative utilization per station) are used for renumbering purposes. Otherwise, the priority values q_j (generalized positional weights), which more directly refer to the precedence structure of the graph, are applied.

Assuming that the stations are built in *forward direction* from the first to the last, the shortly described branching process of SALOME can be applied to ARALBP-1 without any change and is, thus, contained in ABSALOM, too.

In case that no station restrictions are to be considered, the same procedure can be applied in *backward direction* by reversing the precedence graph, i.e., building stations from the last to the first. Furthermore, the **bidirectional branching method** of SALOME can also be used. It consists of systematically alternating forward and backward steps, so that for each node it is decided whether a forward or backward step is taken next. The decision is based on a priority rule following the same ideas as the local renumbering scheme. For details see Scholl and Klein (1997).

If station restrictions are present, backward or bidirectional branching is not useful, because the final station numbers cannot be identified from the backward perspective as long as the resulting total number of stations is not known. Thus, it cannot be decided if a partial solution (partially) generated in backward direction is feasible or not before it is completed by assigning all remaining tasks and finally arranging the stations.

4.3. Bounding

A node of the enumeration tree is fathomed whenever its (local) lower bound LB is equal to or greater than the current upper bound UB (number of stations in the incumbent, i.e., best known solution). The (local) lower bound LB is defined as the maximal value obtained by the bound arguments described below. Bounding is performed for each maximal undominated load (cf. Section 4.4) before the corresponding subproblem is actually build.

The simple **capacity bound** $LB1$ can be extended to $LB1'$ by additionally considering the task attributes as follows:

$$LB1' = \left\lceil \max\left\{\frac{u^r_{sum}}{A_r} \,\middle|\, r \in R_0\right\} \right\rceil \quad \text{with} \quad u^r_{sum} = \sum\nolimits_{j=1}^{n} u_{jr} \text{ for } r \in R \text{ and } u^0_{sum} = t_{sum} \tag{17}$$

Each resource including the time as basic resource $r = 0$ defines an own bound $LB1$, the largest of which is used as $LB1'$. If the bound is to be applied to a residual problem, the remaining sums based on unassigned tasks only are to be considered. This bound argument is equivalent to the capacity bound for resource constraint project scheduling (Klein and Scholl 1999).

The **counting bounds** LB2, LB3 and LB6 contained in SALOME (cf. Scholl 1999, ch. 2.2.2.1) can be modified in a similar manner by computing a bound value for each resource $r \in R_0$ and taking the maximal value which is then rounded up if necessary. For LB2, the resource-dependent values are computed as the number of tasks with $u_{jr} > A_r / 2$ plus half of the number of tasks with $u_{jr} = A_r / 2$ for $r \in R_0$. Additionally, LB2 can be increased by adding 1 for each task which is incompatible to all tasks considered in the computation of LB2 before. LB3 represents a more detailed view with respect to thirds of the resource supply (for details of both bound types see Johnson 1988, Scholl and Klein 1999). The logic behind these simple counting bounds is combined and extended to form a more sophisticated bound LB6 (cf. Berger et al. 1992).

The **one-machine bound** LB4 is based on heads and tails, the computation of which might be strengthened due to the different resources (Johnson 1988, Scholl and Klein 1999). Since computing the head of a task requires to know the heads of all predecessors in a recursive manner (see below), the heads have to be computed following a topological ordering. Equivalently, tails are computed in a reverse topological order.

A *head* a_{jr} can be computed for each resource $r \in R_0$ by applying the logics of LB1, LB2, and LB3 to the predecessor set P_j^* as described by Scholl (1999, ch. 2.2.2.1) for the resource type time by replacing (relative) durations by (relative) resource usages τ_{jr}. Additionally, a one-machine problem can be solved for each r with P_j^* defining the set of jobs to be scheduled. The jobs are sorted according to non-decreasing heads and the minimal makespan obtained for this list of jobs serves as a possible head for task j. The maximum of these (still unrounded) single head values for resource r serves as the initial value of the head a_{jr}. Non-integral values a_{jr} may be strengthened whenever task j does not fit into the last of the $\lceil a_{jr} \rceil$ first stations, i.e., $a_{jr} + \tau_{jr} > \lceil a_{jr} \rceil$. In such a case, a_{jr} is rounded up to the next integer $\lceil a_{jr} \rceil$.
Obviously, task j cannot be assigned to the first $\bar{a}_j := \max\{\lfloor a_{jr} \rfloor \mid r \in R_0\}$ stations, because its predecessors occupy this number of stations completely with regard to at least one resource r. The *earliest station* to which task j can be assigned feasibly then is given by:

$$E_j := \min\{k \in FS_j \mid k > \bar{a}_j\} \tag{18}$$

Because the resource-individual heads are required in computing heads of follower tasks by the one-machine problem recursively, each value a_{jr} smaller than $E_j - 1$ is finally raised to this value, i.e., $a_{jr} := \max\{a_{jr}, E_j - 1\}$.

Tails n_j are computed in an analogous manner by applying the method for computing heads to the reverse precedence graph or by considering the successor sets F_j^* instead of the predecessor sets P_j^*, respectively. As results, we get tails n_{jr} for each task j and resource $r \in R_0$ and the maximal integer tails $\bar{n}_j$ (lower bound on the number of stations following the one task j is assigned to).

Let m be a trial number of stations. Then, the latest station for task j, provided that at most m stations are available, is computed as:

$$L_j(m) := \max\{k \in FS_j \mid k \le m - \bar{n}_j\} \tag{19}$$

To compute a lower bound LB4 to the overall problem, a fictitious sink node n+1 with $\tau_{n+1,r} = 0$ for all $r \in R_0$ and connected with all original sinks and a fictitious source node 0 with $\tau_{0r} = 0$ for all r and connected with all original source nodes are introduced. For each resource $r \in R_0$, a lower bound is computed as follows, based on a makespan minimizing one-machine problem with heads and tails (cf. Carlier 1982):

$$LB_{4r} := \max\{\lceil a_{n+1,r}\rceil, \lceil n_{0r}\rceil, \max\{\lceil a_{jr} + \tau_{jr} + n_{jr}\rceil \mid j = 1, \ldots, n\}\} \quad \text{for } r \in R_0 \tag{20}$$

The maximal of those bound values serves as LB4: (21)

$$LB_4 := \max\{LB_{4r} \mid r \in R_0\} \tag{22}$$

Since LB4 is computationally more expensive than LB_2 and LB_3, this bound is only determined in the root node. In all other nodes, earliest and latest stations are computed based on the logic of LB1 as follows. Additionally, it is accounted for that no task must be assigned to an earlier / later station than one of its predecessors / successors.

$$E'_j := \min\left\{k \in FS_j \mid k \geq \max\left\{\left\lceil \tau_{jr} + \sum\nolimits_{i \in P_j^*} \tau_{ir} \right\rceil \middle| r \in R_0\right\} \wedge k \geq E_i \quad \forall\, i \in P_j^*\right\} \tag{23}$$

$$L'_j(m) := \max\left\{k \in FS_j \mid k \leq m + 1 - \max\left\{\left\lceil \tau_{jr} + \sum\nolimits_{i \in F_j^*} \tau_{ir} \right\rceil \middle| r \in R_0\right\} \wedge k \leq L_i \;\forall\, i \in F_j^*\right\} \tag{24}$$

Based on earliest and latest stations (computed by (18) and (19) or (23) and (24)), the **feasible stations bound** LB5 (cf. Scholl and Becker 2006) is defined as the smallest number of stations for which each task has at least one feasible station:

$$LB5 := \min\{m \in \mathbb{Z}^+ \mid L_j(m) \geq E_j \quad \forall\, j \in V\} \tag{25}$$

Finally, the **SALBP-2 based bound** LB7 can also be applied to ARALBP-1 by extending the underlying bounds on the cycle time (resource time) to all resource types. For each trial value m, a lower bound $\underline{c}(m)$ on the cycle time is computed by a (simple) bound argument for SALBP-2. If $\underline{c}(m) > c$ holds, no feasible solution with at most m stations can exist. Thus, m is increased by 1 and the process is repeated. Otherwise, m constitues a first lower bound on the number of stations. Starting with the current value of m, the same logic is applied to all other resources $r \in R$ one after the other. In each case, a lower bound $\underline{A}_r(m)$ on the required amount of resource r per station is computed by adapting the SALBP-2 bound. If $\underline{A}_r(m) > A_r$ holds, m is increased and a new bound $\underline{A}_r(m)$ is computed. Otherwise, the next resource r is considered. After having examined all resources, the current value of m constitutes the lower bound LB7. As bound argument for SALBP-2 we use the one originally proposed by Scholl and Klein (1997).

Both, LB5 and LB7, are based on the so-called *destructive improvement*, a technique from constrained programming originally applied to the resource constrained project scheduling problem (cf. Klein and Scholl 1999). Trial numbers of stations are contradicted (destroyed) by domain reduction and other logical tests.

Of course, further bound arguments, including the one proposed by Fleszar and Hindi (2003), could be integrated (see the survey in Scholl and Becker 2006). However, we restrict ourselves to the bound arguments originally contained in SALOME in order to be comparable to former studies.

4.4. Logical tests

SALOME contains a number of locical tests, namely, the maximum load rule, the task time incrementing rule, the Jackson dominance rule, the tree dominance rule and dynamic prefixing (cf. Scholl and Klein 1997, 1999; Scholl 1999, ch. 3.3.2.3). These rules combine approaches from combinatorial optimization and constraint programming.

Maximum load rule (MLR)

A feasible load S_k for a station k (based on known loads for stations $1,...,k-1$) is defined to be *maximal* if it is not possible to assign an additional, yet unassigned task in a feasible manner. When solving SALBP-1, a load is maximal if no task can be added without violating the precedence constraints and exceeding the cycle time. In case of ARALBP-1, the definition of load maximality is to be extended to additionally observe station restrictions, task incompatibilities and resource restrictions. As a consequence, loads which are non-maximal with respect to SALBP-1 might be maximal for ARALBP-1.

For SALBP-1, it is known that at least one optimal solution consists of maximal loads only and, thus, only maximal loads must be considered for branching (Jackson 1956). The same is obviously true for ARALBP-1, because no advantage with respect to the resulting number of stations can be obtained by delaying an assignable task, i.e. assigning it to a later station and leaving the respective capacity of the current station unused. By the way of contrast, maximal task distances would require to deactivate the MLR, because observing the distances might require to delay tasks. Besides practical irrelevance, this was a reason to exclude maximal distances.

(Extended) Task Time Incrementing Rule (TTIR)

It is used prior to enumeration in the root node to increase times t_j and, by analogy, resource requirements u_{jr} of some tasks j in order to improve bound computations, if possible.

In its most simple form for SALBP-1, the TTIR is used to raise the time t_j of a task j to c if the condition $t_j + t_{min} > c$ is fulfilled with $t_{min} = \min\{t_h \mid h \in V\}$, because this condition ensures that task j cannot be combined with any other task in the same station (cf. Johnson 1988).

We apply and extend a more versatile version of the rule (see Scholl 1999, ch. 4.1.2.3) which examines more directly whether or not a task can be combined with any other one. In order to do so, the following sets are computed:

$$FP_{hj} := FP_{jh} := (F_h^* \cap P_j^*) \cup (F_j^* \cap P_h^*) \qquad \text{for } h<j \text{ with } h,j \in V \qquad (26)$$

$$CT_j := \left\{ h \in V \mid FS_j \cap FS_h \neq \{\ \} \wedge (h,j) \notin IT \wedge u_{jr} + u_{hr} + \sum\nolimits_{i \in FP_{hj}} u_{ir} \leq A_r \ \forall r \in R_0 \right\} \qquad (27)$$

Considering a pair (h,j) of tasks, FP_{hj} is defined to contain all tasks lying on paths connecting h and j or j and h, respectively. The set CT_j collects all tasks which are potentially combinable with task $j \in V$. Combinable tasks (1) must have at least one feasible station in common and (2) must not be incompatible and (3) must not exceed the resource capacities together with all intermediate tasks. If the set CT_j is empty, no task is potentially combinable with j and the resource requirements can be set to the capacities for all resources, i.e., $u_{jr} := A_r$ for all $r \in R_0$, which includes raising the task time $t_j = u_{j0}$ to $A_0 = c$.

After having successfully increased particular resoure requirements, lower bounds might be improved (e.g., LB1 due to increased u^r_{sum} values) and enumeration will usually be accelerated.

Jackson Dominance Rule (JDR)

The JDR (Jackson 1956, strengthened by Scholl and Klein 1997) is applied to reduce the number of alternative loads which have to be considered for a certain station k. It is based on potential dominance. In case of SALBP-1, a task h *potentially dominates* another task j that is not related to h by precedence, if $F_j \subseteq F_h^*$ and $t_j \le t_h$ hold.

The rule is applied (in forward direction) to exclude a maximal station load S_k from being branched if at least one task $j \in S_k$ can be replaced feasibly by a still unassigned task h which potentially dominates j. The conditions of potential dominance ensure that each complete solution which contains the excluded load, i.e., $j \in S_k$ and $h \in S_q$ with some station $q > k$, can be transformed into an equivalent feasible solution with j and h exchanged. This is due to the fact that all successors of task j are successors of h as well ($F_j \subseteq F_h^*$) and, thus, cannot start before h is finished. Hence, the sequence of j and h is not important for the successors of j. Together with $t_j \le t_h$ it is ensured that replacing h by j is feasible in S_q. So, it is sufficient to complete the dominating (feasible) load in station k.

For ARALBP-1, the definition of the potential dominance is to be extended. Now, the following conditions must be fulfilled to constitute a potential dominance of task h over task j:

- h and j are not related by precedence and $F_j \subseteq F_h^*$ (see above)
- $u_{jr} \le u_{hr}$ for all $r \in R_0$ — task j can replace task h in S_q concerning *all* resources
- $FS_h \subseteq FS_j$ — task j can replace task h in S_q concerning station restrictions
- $IT_j \subseteq IT_h$ — task j can replace task h in S_q concerning incompatibilities

If all conditions are fulfilled as equations, the lower-numbered task is defined to be dominating.

The JDR is also applied in the extended version proposed by Scholl and Klein (1999), where a load S_k is excluded due to an unassigned task h which can replace a subset $J \subseteq S_k$ feasibly and potentially dominates each $j \in J$. Additionally, the conditions $u_{hr} \ge \sum_{j \in J} u_{jr}$ must be fulfilled for each $r \in R_0$ to ensure that h can be replaced by the complete set J in S_q.

Modification of the Task Renumbering. In order to find feasible solutions as fast as possible, the task renumbering explained in Section 4.2 is modified according to potential dominances. By giving each task j which is potentially dominated by a task h a larger number than h, dominating station loads are built prior to dominated ones.

Tree dominance rule (TDR)

The TDR identifies partial solutions which contain the same set of tasks than another, previously enumerated one. A maximal undominated load S_k is excluded if the set of tasks currently assigned to $S_1, ..., S_k$ have already been assigned to at most k stations in an earlier step. Thus, this rule necessitates to store the station needs for each already assigned subset of tasks. An memory-saving and easy-to-access method is based on a tree structure introduced by Nourie and Venta (1991). The TDR can be applied to ARALBP-1 without modifications as it is a straight-forward extension of the MLR to more than one stations (provided that no maximal task distances are considered).

Dynamic prefixing (DP)

DP is a reduction rule directly exploiting the concept of earliest and latest stations (cf. Section 3) and can be viewed as avoiding predictable bounding in later stages of the following subtree. Before finding station loads of a current station k by enumeration, prefixing is used to assign tasks which cannot be assigned to a later station than k.

In case of SALBP-1, a *temporary prefixing* is performed with respect to the current local lower bound LLB, i.e., a task j is (temporarily) prefixed to the currently considered station k if $L_j(LLB) = k$. The prefixed tasks remain in station k until the local lower bound LLB_k is increased. Due to the equation $L_j(m+1) = L_j(m)+1$, no new temporary prefixing can occur in station k for the increased local bound. This technique is analogously applicable in backward direction based on earliest stations E_j. In either case it also applies to ARALBP-1.

Additionally, the station restrictions may enforce a *final prefixing* for the current node of the tree. This is the case if the current station k is the largest station index contained in the set of feasible stations FS_j for a task j. Then, this task must be assigned to k to obtain a feasible solution at all, irrespective of the current value of LLB. If this is not possible for such a task due to capacity restrictions or incompatibilities, the current node of the tree needs to be fathomed.

5. Computational experiments

We describe a number of computational experiments and analyse their results to examine the performance of ABSALOM.

5.1. Data sets

Up to now, there are no systematic data sets available for ARALBP-1. As a consequence, we generate a number of data sets by systematically varying the influence of different types of assignment restrictions. As a basis, we use the well-known benchmark data set for SALBP-1 (cf. Scholl 1999, ch. 7.1; available for download at the homepage for assembly line optimization research: *www.assembly-line-balancing.de*). This data set consist of 269 instances, based on 25 precedence graphs having 8 to 297 nodes each connected to several cycle times. Three configurations have been used to generate data sets.

Setting 1: Data sets for task and resource restrictions

In this setting, the task and resource assignment restrictions $\alpha_5 \in \{\text{link}, \text{inc}, \text{cum}\}^*$ are considered and variied in a full factorial manner. The following parameters are used and variied in the given sets (parameter values in parantheses are only considered in individual tests).

$\lvert R\rvert$	number of resources; $\lvert R\rvert \in \{0, 1, (2), 3\}$
LR	ratio of linked task pairs (in %); $LR \in \{0, (2), 5, (7), 10, (20)\}$
IR	ratio of incompatible task pairs (in %); $IR \in \{0, (2), 5, (7), 10, (20)\}$

For each combination $(|R|, LR, IR)$ tested, a data set with 269 instances is generated from the benchmark set for SALBP-1. For each instance, it is ensured that the optimal and known SALBP-1 solution remains feasible and, thus, optimal. This gives opportunity to test the performance of algorithms on the best possible basis.

Consider a SALBP-1 instance and a corresponding optimal solution given as a sequence of station loads $S_1, S_2, \ldots S_{m*}$ and by assignment variables z_j for $j \in V$ which specify the index of the station task j is assigned to, i.e. $z_j = k$ if $j \in S_k$. In order to maintain feasibility and optimality of the SALBP-1 solution , the data parameters required are set as follows:

- *Resource restrictions:* For each resource $r \in R$, the capacity is set to $A_r = 100$ (normalization). Each station gets these initial capacities as $A_{rk} = 100$ for $r \in R$ and $k \in \{1, \ldots, m*\}$. For each $r \in R$, the tasks j are considered in order of their numbering. The resource requirement u_{jr} is randomly chosen from the interval $[0, A_{rk}]$ with $k = z_j$ using uniformly distributed random numbers. Afterwards, the station capacity is reduced to $A_{rk} := A_{rk} - u_{jr}$. This process is repeated until all tasks have been considered. If $A_{rk} = 0$ is reached for some station k, the possible remaining tasks in S_k get zero requirements.
- *Linked tasks:* Each pair of tasks (i, j) both of which are assigned to the *same* station, i.e., $z_i = z_j$, is considered and defined as a linked pair with given probability LR based on uniformly distributed random numbers.
- *Incompatible tasks:* The pairs (i, j) of tasks assigned to *different* stations in the SALBP-1 solution, are defined to be incompatible with given probability IR also using uniformly distributed random numbers.

A first collection of data sets is obtained by considering the restriction types separately. In each case all parameter values given above are examined. In order to restrict the total effort, a second collection of data sets is obtained by combining all parameter values not set in parantheses. This amounts to 34 data sets each of which consists of 269 instances, i.e., a total of 9,146 instances (see Table 3).

Setting 2: Data sets including station restrictions

Because considering station restrictions requires ABSALOM to be restricted to forward direction, these restrictions are added separately. An additional parameter is used:

SR ratio of restricted feasible station sets (in %); $SR \in \{0, 25, 50, 75, 100\}$

Initially, the earliest and latest stations are defined as $E_j := 1$ and $L_j := \overline{m}$ for all $j \in V$, where $\overline{m} = 1.5 \cdot m*$ serves as an arbitrarily chosen upper bound on the number of stations. The tasks are selected with probability SR. For each selected task j the station interval is randomly reduced as follows: E_j is set to a value in $[1, z_j]$, while L_j gets a value in $[z_j, \overline{m}]$ using uniformly distributed random numbers. If $E_j = L_j = z_j$ results, the situation of a fixed station ($\alpha_5 = \text{fix}$) is present. In other cases, $\alpha_5 = \text{type}$ is reflected by restricting the task to a certain segment of the line. Finally, 10% of the station indices in the resulting sets $FS_j = \{E_j, \ldots, L_j\}$ are removed randomly to reflect $\alpha_5 = \text{excl}$.

The different values of SR are combined with two selected combinations $(|R|, LR, IR) = (0,0,0)$ and $(|R|, LR, IR) = (3,5,5)$ to consider station restrictions alone and combined with all other types of assignment restrictions. This results in 10 data sets (2,690 instances; see Table 4).

Setting 3: Data set for time- and space- constrained SALBP-1

Bautista and Pereira (2007) consider a time- and space-constrained problem, which consists of SALBP-1 extended by a single resource constraint interpreted as limited space ($|R| = 1$). No further assignment restrictions are considered. The data set also uses the benchmark set for SALBP-1 and generates the data for resource $r = 1$ as follows:

$$u_{j1} = t_{n-j+1} = u_{n-j+1,0} \text{ for all } j \in V \text{ and } A_1 = A_0 = c \qquad (28)$$

5.2. Results for test setting 1

Up to now, there does not exist another exact solution procedure for ARALBP-1. Thus, we use the flexible heuristic AVALANCHE of Boysen and Fliedner (2008) as a benchmark procedure. This heuristic is a two-stage method using an ant colony system heuristic at the first stage and a shortest-path computation at the second stage. It is the only procedure flexible enough to cope with many extensions of assembly line balancing problems simultaneously.

The experiments were run on a personal computer with an Intel Pentium IV processor of 1,5 GHz clock speed and 512 MByte of RAM. For each instance a time limit of 500 sec. was imposed on ABSALOM, while AVALANCHE is configured as follows: Linked tasks are merged to mega nodes prior to applying the procedure. In each run of AVALANCHE, 1000 iterations with a population size of 25 sequence vectors (ants) are performed.

The following measures are used to evaluate the algorithms:

- # opt.: number of proven optima found (out of 269 instances in each data set)
- # found: number of optima found (the optimal solution is retrieved, i.e., the final UB is equal to m*, but could not necessarily be proven within the given time span)
- rel. dev.: average relative deviation from minimal number of stations (or best known lower bound if the optimum is still unknown which is the case for one of the 269 instances)
- total time: average computation time (with values of 500 sec. for time outs of ABSALOM)

Table 3 summarizes the results. Obviously, ABSALOM clearly outperforms AVALANCHE with respect to all measures. It finds much more optimal solutions (86% vs. 35%; see the summarizing row 34), the relative deviations from optimum are better by an order of magnitude (1% vs. 11% on average) and even computation times are comparable (80 vs. 46 sec. on average). Furthermore, a considerable number of proven optimal solutions is obtained (85%), which is not achievable by the heuristic procedure AVALANCHE unless it is supplemented with bounding procedures.

Considering isolated variations of the three parameters shows that the number of resource constraints $|R|$ has the greatest impact on the procedures' performance (rows 0 to 3). In both cases, the different measures deteriorate considerably.

A milder increase in complexity and loss in solution quality is obtained by incompatibility constraints (rows 9 to 13). These constraints make the assignment of tasks to stations more sensitive,

Table 3. Results of ABSALOM and AVALANCHE for setting 1

					ABSALOM (bidirectional)				AVALANCHE		
row no.	problem type	|R|	LR	LR	# opt.	# found	rel. dev.	total time	# found	rel. dev.	total time
0	SALBP-1	0	0	0	254	262	0,06%	30,80	162	2,19%	63,55
1	resource constraints only	1	0	0	225	229	0,82%	85,20	67	11,85%	58,03
2		2	0	0	200	202	2,04%	133,21	53	16,80%	59,56
3		3	0	0	202	202	2,79%	130,36	51	19,40%	61,57
4	linked tasks only	0	2	0	250	257	0,10%	37,74	157	2,51%	54,60
5		0	5	0	252	257	0,11%	35,43	159	2,48%	46,23
6		0	7	0	248	254	0,13%	40,15	158	2,40%	40,31
7		0	10	0	247	252	0,17%	44,78	152	2,76%	34,82
8		0	20	0	250	254	0,15%	35,90	160	2,46%	24,86
9	incompatible tasks only	0	0	2	256	262	0,22%	30,13	135	3,85%	51,40
10		0	0	5	250	256	0,42%	39,82	115	6,29%	55,77
11		0	0	7	247	253	0,45%	46,86	97	8,28%	56,76
12		0	0	10	239	245	0,76%	64,44	94	11,05%	57,01
13		0	0	20	224	230	1,38%	88,42	79	19,27%	59,03
14	linked and incompatible tasks	0	5	5	245	250	0,23%	49,68	121	5,56%	41,00
15		0	5	10	243	249	0,29%	51,28	101	8,13%	45,00
16		0	10	5	248	254	0,14%	43,82	125	4,64%	35,91
17		0	10	10	239	245	0,26%	60,76	116	6,28%	40,19
18	a single resource constraint	1	0	5	214	216	1,18%	107,52	63	16,09%	55,93
19		1	5	0	228	231	0,72%	82,58	71	9,55%	40,38
20		1	5	5	224	226	0,75%	88,44	64	12,51%	37,59
21		1	0	10	204	206	1,68%	123,51	60	19,88%	57,45
22		1	10	0	230	233	0,60%	78,54	86	8,12%	30,10
23		1	5	10	218	220	0,93%	100,08	71	14,57%	41,06
24		1	10	5	226	229	0,75%	82,68	81	9,78%	29,21
25		1	10	10	226	228	0,80%	83,04	89	11,00%	30,97
26	several resource constraints	3	0	5	197	197	3,16%	139,89	52	22,64%	56,18
27		3	5	0	207	207	2,24%	120,83	59	13,79%	48,99
28		3	5	5	215	213	2,20%	107,85	62	15,81%	42,49
29		3	0	10	197	195	3,39%	143,30	55	25,47%	56,63
30		3	10	0	217	219	1,83%	99,91	72	11,92%	37,95
31		3	5	10	218	220	1,84%	99,58	81	12,95%	34,34
32		3	10	5	213	213	2,15%	113,31	65	17,64%	46,09
33		3	10	10	220	222	1,80%	97,38	79	13,59%	35,33
34	all	total			7773 (84,99%)	7888 (86,25%)	1,07%	79,92	3212 (35,12%)	10,93%	46,07

because less unassigned tasks may remain which can be combined to form favourable loads. By the way of contrast, the portion of linked tasks has only minor influence on the algorithmic performance, respectively (rows 4 to 8). Reduction in problem size by merging tasks and increased complexity of combining larger tasks seem to counterbalance each other. When combined, linked tasks attenuate the influence of incompatibilities on solution quality (e.g., see rows 5, 10, and 14 or 5, 12, and 15). The same effect is to be observed in combination with one (rows 18 to 25) or three resource constraints (rows 26 to 33) and valid for both procedures.

To summarize, ABSALOM turns out be an effective procedure for solving balancing problems with task and resource restrictions, while AVALANCHE is not competitive. This is due to the fact that the latter procedure is a multi-purpose solution approach not specially designed for a specific problem configuration.

5.3. Results for test setting 2

For setting 2, AVALANCHE is not applicable, because its approach to solving the problem requires data that is independent from the actual position of a station on the line. ABSALOM must be applied as a forward procedure, because the final station position is necessary for examining station restrictions but not known when backward branching is performed. So, we only give results of the forward version of ABSALOM.

Table 4. Results of ABSALOM for setting 2

					ABSALOM (forward)			
row no.	|R|	LR	LR	SR	# opt.	# found	rel. dev.	total time
0	0	0	0	0	231	218	0,46%	109,22
1	0	0	0	25	228	219	0,49%	104,89
2	0	0	0	50	229	219	0,51%	101,67
3	0	0	0	75	228	219	0,57%	101,99
4	0	0	0	100	218	211	0,74%	109,87
5	3	5	5	0	191	194	2,91%	152,08
6	3	5	5	25	193	193	2,94%	154,38
7	3	5	5	50	183	188	3,66%	164,16
8	3	5	5	75	188	188	3,99%	160,65
9	3	5	5	100	185	186	4,68%	161,18

Table 4 summarizes the results obtained for variied ratio of station restrictions SR with two settings for the other parameters. Comparing the performance of the forward procedure with the bidirectional one for SALBP-1 (rows 0 in Table 3 and Table 4) reveals that the forward procedure is considerably less effective as has already been stated by Scholl and Klein (1997).

With respect to the station restrictions, the results are quite surprising: The smaller the feasible station sets FS_j (increasing SR), the more complex the problems seem to be. One could have expected the opposite, because shrinking station sets leads to reduced solution spaces. However, it seems to be more complex to find favourable task combinations when sets are small as is the case with increasing incompatibilies (see Section 5.2). This dependency between SR and solution quality is found for both settings of the remaining parameters even though at different quality levels.

5.4. Results for test setting 3

Finally, we compare ABSALOM and AVALANCHE with the best ant algorithm ANTS developed by Bautista and Pereira (2007) for the time- and space-constrained SALBP-1. In order to be comparable, the time limit is set to 30 sec. for ABSALOM (for ANTS Bautista and Pereira set a limit of 120 sec. but used a slower computer). AVALANCHE is used without a time limit because it is not a specialized procedure with some overhead due to generality.

Table 4 summarizes the results. ABSALOM finds proven optimal solutions for 183 instances (68%). Since only this part of optimal solutions is known, the relative deviations are computed based on the best known lower bounds (equal to the optimal value if known). As a consequence the average relative deviation of 3.57% is an upper bound on the true average deviation which should be fairly smaller.

Table 5. Results of ABSALOM, AVALANCHE and ANTS for setting 3

ABSALOM					AVALANCHE			ANTS	
# opt.	rel. dev.	# found	rel. dev.	total time	# found	rel. dev.	total time	# found	rel. dev.
183	3.57%	66	9.19%	10.54	25	18.13%	62.73	52	11.06%
related to ARALBP-1		related to SALBP-1			related to SALBP-1			related to SALBP-1	

In order to compare the results to those given by Bautista and Pereira (2007), we additionally use the optimal SALBP-1 solutions as reference points. Due to the additional resource constraint, a SALBP-1 solution is usually not feasible for the resulting ARALBP-1 instance. However, the number of stations in the optimal SALBP-1 solution serves as a (rather weak) lower bound. The measures "# found" and "rel.dev." refer to these SALBP-1 based bounds. On this basis, ABSALOM outperforms ANTS, because it finds the same number of stations as in the SALBP-1 solution (which is sufficient but not necessary for being optimal for ARALBP-1; see above) for 66 instances while this is achieved by ANTS for only 52 instances. The average relative deviation from SALBP-1 bounds is considerably smaller for ABSALOM. This is especially remarkable since ABSALOM is designed as an exact solution procedure and thus needs to use most of its computational resources to examine all branches of the enumeration tree (i.e., the complete solution space) explicitly or implicitly, while ANTS is a specialized heuristic for just this problem setting which can concentrate on promising parts of the solution space.

Though AVALANCHE takes significantly longer computation times it is not competitive. This is partly due to its flexibility overhead as mentioned before and shows the need for specialized procedures.

6. Conclusions and future research

In this paper, we have introduced and modelled the assembly line balancing problem with assignment restrictions (ARALBP-1). For solving this problem, the branch-and-bound procedure ABSALOM has been developed, which is an extension of the well-known procedure SALOME for the simple assembly line balancing problem (SALBP-1). Comprehensive computational experiments based on extended benchmark data sets indicate that the new procedure is effective in finding optimal or at least near-optimal solutions.

The inclusion of assignment restrictions into the successful solution approach of SALOME contributes to closing the gap between assembly line optimization research and practice (Boysen et al. 2008), because such restrictions are relevant in almost any real-world problem setting. Furthermore, more complex real-world problem extensions might be tackled by flexible search procedures based on the repeated solution of ARALBP-1 instances by ABSALOM (for such an approach based on repeated solution of SALBP-1 instances, see Scholl et al. 2006).

For example, assignment restrictions are obviously helpful in modelling and solving ALBPs for lines with large workpieces such as automobiles or lorries where only a part of the mounting positions can be reached within the same station or workplace due to distance, access or position restrictions as is the case with, among others, *two-sided lines*. Once defined, due to actual conditions on the line or within a search process as mentioned above, such restrictions can be modelled by incompatibility constraints and station restrictions. Similarly, assignment restrictions might be useful for ALBP extensions such as selecting the station equipments (e.g., tasks that require different machines or tools which cannot be located in the same station are to be set incompatible) or positioning material boxes at the stations (restricted storage space might be modelled as resource restriction).

Future research should identify and analyze these and further problem extensions that might be solved based on assignment restrictions and develop ARALBP-1 based search procedures (using an exact procedure like ABSALOM or heuristics for ARALBP-1) in order to be able to solve more real-world problems in a satisfactory manner than it is possible up to now.

References

Agnetis, A., A. Ciancimino, M. Lucertini, M. Pizzichella. 1995. Balancing flexible lines for car components assembly. *Internat. J. Product. Res.* **33** 333–350.

Arcus, A.L. 1966. COMSOAL: A computer method of sequencing operations for assembly lines. *Internat. J. Product. Res.* **4** 259–277.

Bartholdi, J.J. 1993. Balancing two-sided assembly lines: A case study. *Internat. J. Product. Res.* **31** 2447–2461.

Bautista, J., J. Pereira. 2002. Ant algorithms for assembly line balancing. *Lecture Notes in Comp. Sci.* **2463** 65–75.

Bautista, J., J. Pereira. 2006. Ant algorithms for a time and space constrained assembly line balancing problem. *European J. Oper. Res.* **177** 2016-2032.

Bautista, J., R. Suarez, M. Mateo, R. Companys. 2000. Local search heuristics for the assembly line balancing problem with incompatibilities between tasks. *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, San Francisco, CA, 2404–2409.

Baybars, I. 1986. A survey of exact algorithms for the simple assembly line balancing problem. *Management Sci.* **32** 909-932.

Becker, C., A. Scholl. 2006. A survey on problems and methods in generalized assembly line balancing. *European J. Oper. Res.* **168** 694-715.

Berger, I., J. Bourjolly, G. Laporte. 1992. Branch-and-bound algorithms for the multi-product assembly line balancing problem. *European J. Oper. Res.* **58** 215-222.

Bhattacharjee, T.K., S. Sahu. 1987. A heuristic approach to general assembly line balancing. *Internat. J. Oper. & Product. Management* **8** 67-77.

Bowman, E. 1960. Assembly-line balancing by linear programming. *Oper. Res.* **8** 385-389.

Boysen, N., M. Fliedner, 2008. A versatile algorithm for assembly line balancing. *European J. Oper. Res.* **184** 39-55.

Boysen, N., M. Fliedner, A. Scholl. 2007. A classification of assembly line balancing problems. *European J. Oper. Res.* **183** 674-693.

Boysen, N., M. Fliedner, A. Scholl. 2008. Assembly line balancing: Which model to use when? *Internat. J. Product. Economics* **111** 509-528.

Buxey, G.M. 1974. Assembly line balancing with multiple stations. *Management Sci.* **20** 1010–1021.

Carlier, J. 1982. The one-machine sequencing problem. *European J. Oper. Res.* **11** 42-47.

Carnahan, B.J., B.A. Norman, M.S. Redfern. 2001. Incorporating physical demand criteria into assembly line balancing. *IIE Transactions* **33** 875–887.

Dar-El, E., Y. Rubinovitch. 1979. MUST - A multiple solutions technique for balancing single model assembly lines. *Management Sci.* **25** 1105-1114.

Deckro, R.F. 1989. Balancing cycle time and workstations. *IIE Transactions* **21** 106–111.

Erel, E., S.C. Sarin. 1998. A survey of the assembly line balancing procedures. *Product. Plann. & Contr.* **9** 414-434.

Gadidov, R., W. Wilhelm. 2000. A cutting plane approach for the single-product assembly system design problem. *Internat. J. Product. Res.* **38** 1731–1754.

Ghosh, S., R.J. Gagnon. 1989. A comprehensive literature review and analysis of the design, balancing and scheduling of assembly line systems. *Internat. J. Product. Res.* **27** 637-670.

Gökcen, H., E. Erel. 1997. A goal programming approach to mixed-model assembly line balancing problem. *Internat. J. Product. Economics* **48** 177–185.

Hautsch, K., H. John, H. Schürgers. 1972. Taktabstimmung bei Fließarbeit mit dem Positionswert-Verfahren. *REFA-Nachrichten* **25** 451–464.

Jackson, J.R. 1956. A computing procedure for a line balancing problem. *Management Sci.* **3** 261-271.

Johnson, R.V. 1983. A branch and bound algorithm for assembly line balancing problems with formulation irregularities. *Management Sci.* **29** 1309-1324.

Johnson, R.V. 1988. Optimally balancing large assembly lines with "FABLE". *Management Sci.* **34** 240-253.

Johnson, R.V. 1991. Balancing assembly lines for teams and work groups. *Internat. J. Product. Res.* **29** 1205-1214.

Kim, Y.K., Y. Kim, Y.J. Kim. 2000. Two-sided assembly line balancing: a genetic algorithm approach. *Product. Plann. & Contr.* **11** 44–53.

Kim, H., S. Park. 1995. A strong cutting plane algorithm for the robotic assembly line balancing problem. *Internat. J. Product. Res.* **33** 2311–2323.

Klein, R., A. Scholl. 1999. Computing lower bounds by destructive improvement - An application to resource-constrained project scheduling. *European J. Oper. Res.* **112** 322-346.

Klenke, H. 1977. *Ablaufplanung bei Fließfertigung*. Gabler, Wiesbaden.

Lapierre, S.D., A.B. Ruiz. 2004. Balancing assembly lines: An industrial case study. *J. Oper. Res. Soc.* **55** 589–597.

Lapierre, S.D., A. Ruiz, P. Soriano. 2006. Balancing assembly lines with tabu search. *European J. Oper. Res.* **168** 826–837.

Lee, T.O., Y. Kim, Y.K. Kim. 2001. Two-sided assembly line balancing to maximize work relatedness and slackness. *Comp. & Indust. Engin.* **40** 273–292.

Leu, Y.Y., L.A. Matheson, L.P. Rees. 1994. Assembly line balanc ing using genetic algorithms with heuristicgenerated initial populations and multiple evaluation criteria, *Dec. Sci.* **25** 581-606.

Liu, C.-M., C.-H. Chen. 2002. Multi-section electronic assembly line balancing problems: A case study. *Product. Plann. & Contr.* **13** 451-461.

Malakooti, B., A. Kumar. 1996. A knowledge-based system for solving multi-objective assembly line balancing problems. *Internat. J. Product. Res.* **34** 2533–2552.

Miltenburg, J. 1998. Balancing U-lines in a multiple U-line facility. *European J. Oper. Res.* **109** 1–23.

Miltenburg, J. 2006. Optimally balancing large assembly lines: Updating Johnson´s 1988 FABLE algorithm. *INFOR: Inform. Syst. and Oper. Res.* **44** 23-47.

Miralles, C. 2005. Solving procedures for the assembly line worker assignment and balancing problem: application to sheltered work centres for disabled. *XI Escuela Latinoamericana de Verano en Investigación de Operaciones*, Villa de Leyva, Colombia (http://elavio2005.uniandes.edu.co/ResumenesParticipantes/Lunes/MirallesCristobal_R.pdf).

Nourie, F.J., E.R. Venta. 1991. Finding optimal line balances with OptPack. *Oper. Res. Letters* **10** 165-171.

Park, K., S. Park. W. Kim. 1997. A heuristic for an assembly line balancing problem with incompatibility, range, and partial precedence constraints. *Comp. & Indust. Engin.* **32** 321–332.

Pastor, R., C. Andres. A. Duran, M. Perez. 2002. Tabu search algorithms for an industrial multi-product and multi-objective assembly line balancing problem, with reduction of the task dispersion. *J. Oper. Res. Soc.* **53** 1317–1323.

Pastor, R., A. Corominas. 2000. Assembly line balancing with incompatibilities and bounded workstation loads. *Ricerca Operativa* **30** 23–45.

Pastor, R., A. Corominas, A. Lusa. 2004. Different ways of modelling and solving precedence and incompatibility constraints in the assembly line balancing problem. *Frontiers in Artificial Intelligence and Applications* **113** 359-366.

Patterson, J.H., J.J. Albracht. 1975. Assembly-line balancing: Zero-one programming with Fibonacci search. *Oper. Res.* **23** 166 - 172.

Pinnoi, A., W.E. Wilhelm. 1997. A family of hierarchical models for assembly system design. *Internat. J. Product. Res.* **35** 253–280.

Raouf, A., C. Tsui. 1982. A new method for assembly line balancing having stochastic work elements. *Comp. & Indust. Engin.* **6** 131–148.

Rekiek, B., P. de Lit, F. Pellichero, T. L'Eglise, P. Fouda, E. Falkenauer, A. Delchambre. 2001. A multiple objective grouping genetic algorithm for assembly line design. *J. Intell. Manufact.* **12** 467–485.

Rekiek, B., P. de Lit, A. Delchambre. 2002. Hybrid assembly line design and user's preferences. *Internat. J. Product. Res.* **40** 1095–1111.

Roberts, S.D., C.D. Villa. 1970. On a multiproduct assembly line-balancing problem. *AIIE Trans.* **2** 361–364.

Saltzman, M.J., I. Baybars. 1987. A two-process implicit enumeration algorithm for the simple assembly line balancing problem. *European J. Oper. Res.* **32** 118-129.

Sawik, T. 2002. Monolithic vs. hierarchical balancing and scheduling of a flexible assembly line. *European J. Oper. Res.* **143** 115–124.

Schofield, N.A. 1979. Assembly line balancing and the application of computer techniques. *Comp. & Indust. Engin.* **3** 53-59.

Scholl, A. 1999. *Balancing and sequencing assembly lines*. 2$^{\text{nd}}$ edn. Physica, Heidelberg.

Scholl, A., C. Becker. 2006. State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European J. Oper. Res.* **168** 666-693.

Scholl, A., N. Boysen, M. Fliedner. 2006. The sequence-dependent assembly line balancing problem. *OR Spectrum* (to appear).

Scholl, A., R. Klein. 1997. SALOME: A bidirectional branch and bound procedure for assembly line balancing. *Informs J. Computing* **9** 319-334.

Scholl, A., R. Klein. 1999. Balancing assembly lines effectively - A computational comparison. *European J. Oper. Res.* **114** 50-58.

Sprecher, A. 1999. A competitive branch-and-bound algorithm for the simple assembly line balancing problem. *Internat. J. Product. Res.* **37** 1787–1816.

Thangavelu, S.R., C.M. Shetty. 1971. Assembly line balancing by zero-one integer programming. *AIIE Trans.* **3** 61-68.

Vilarinho, P.M., A.S. Simaria. 2002. A two-stage heuristic method for balancing mixed-model assembly lines with parallel workstations. *Internat. J. Product. Res.* **40** 1405–1420.

Vilarinho, P.M., A.S. Simaria. 2006. ANTBAL: An ant colony optimization algorithm for balancing mixed-model assembly lines with parallel workstations. *Internat. J. Product. Res.* **44** 291-303.

Wee, T.S., M.J. Magazine. 1982. Assembly line balancing as generalized bin packing. *Oper. Res. Letters* **1/2** 56–58.

White, W.W. 1961. Comments on a paper by Bowman. *Oper. Res.* **9** 274-276.

Wilhelm, W.E., R. Gadidov. 2004. A branch-and-cut approach for a generic multiple-product, assembly-system design problem. *Informs J. Computing* **16** 39-55.

Zäpfel, G. 1975. *Ausgewählte fertigungswirtschaftliche Optimierungsprobleme von Fließfertigungssystemen*. Beuth, Berlin.