Available online at www.sciencedirect.com
SCIENCE DIRECT®
European Journal of Operational Research 168 (2006) 666–693
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/ejor

Invited Review

State-of-the-art exact and heuristic solution procedures for simple assembly line balancing

Armin Scholl *, Christian Becker

Fakultät für Wirtschaftswissenschaften, Friedrich-Schiller-Universität Jena, Carl-Zeiß-Straße 3, D-07743 Jena, Germany

Available online 11 September 2004

Abstract

The assembly line balancing problem arises and has to be solved when an assembly line has to be configured or redesigned. It consists of distributing the total workload for manufacturing any unit of the product to be assembled among the work stations along the line. The so-called simple assembly line balancing problem (SALBP), a basic version of the general problem, has attracted attention of researchers and practitioners of operations research for almost half a century.

In this paper, we give an up-to-date and comprehensive survey of SALBP research with a special emphasis on recent outstanding and guiding contributions to the field.

Keywords: Assembly line balancing; Mass-production; Literature survey; Combinatorial optimization; Branch-and-bound; Heuristics

1. Introduction

Assembly lines are flow-oriented production systems which are still 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.

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

* Corresponding author. Fax: +49 3641 943171.
E-mail addresses: a.scholl@wiwi.uni-jena.de (A. Scholl), c.becker@wiwi.uni-jena.de (C. Becker).

0377-2217/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2004.07.022

the stations with respect to some objective is known as the *assembly line balancing problem* (*ALBP*).

Manufacturing a product on an assembly line requires partitioning the total amount of work into a set of elementary operations named *tasks* $V = \{1,\ldots,n\}$. Performing a task j takes a *task time* t_j and requires certain equipment of machines and/or skills of workers. Due to technological and organizational conditions *precedence constraints* between the tasks have to be observed.

These elements can be summarized and visualized by a *precedence graph*. It contains a node for each task, node weights for the task times and arcs for the precedence constraints. Fig. 1 shows a precedence graph with $n = 10$ tasks having task times between 2 and 9 (time units). The precedence constraints for, e.g., task 5 express that its processing requires the tasks 1 and 4 (*direct predecessors*) and 3 (*indirect predecessor*) be completed. The other way round, task 5 must be completed before its (direct and indirect) *successors* 6, 8, 9, and 10 can be started.

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

In the following, we consider the simple assembly line balancing problem (SALBP). More general ALBP with additional characteristics like cost objectives, paralleling of stations, equipment selection, and mixed-model production are described and surveyed in Becker and Scholl (this issue).

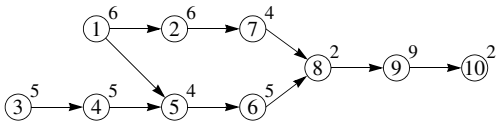

Fig. 1. Precedence graph.

2. Simple assembly line balancing problem (SALBP)

Most of the research in assembly line balancing has been devoted to modelling and solving the *simple assembly line balancing problem* (SALBP). This classical single-model problem contains the following main characteristics (cf. Baybars, 1986a; Scholl, 1999, Chapter 2.2):

- 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* $E = t_{\text{sum}}/(m \cdot c)$ with total task time $t_{\text{sum}} = \sum_{j=1}^{n} t_j$.

Several problem versions arise from varying the objective as shown in Table 1. SALBP-F is a feasibility problem which is to establish whether or not a feasible line balance exists for a given combination of m and c. SALBP-1 and SALBP-2 have a dual relationship, because the first minimizes m given a fixed c, while the second minimizes c (maximizes the production rate) given m. SALBP-E is the most general problem version maximizing the line efficiency thereby simultaneously minimizing c and m considering their interrelationship.

For the example of Fig. 1 with $t_{\text{sum}} = 48$, a feasible line balance for the SALBP-F instance with

Table 1
Versions of SALBP

	Cycle time c	
	Given	Minimize
No. m of stations		
Given	SALBP-F	SALBP-2
Minimize	SALBP-1	SALBP-E

cycle time $c = 9$ and $m = 7$ stations is given by the following sequence of station loads ($S_1 = \{1\}$, $S_2 = \{2\}$, $S_3 = \{3,7\}$, $S_4 = \{4,5\}$, $S_5 = \{6,8\}$, $S_6 = \{9\}$, $S_7 = \{10\}$). While no idle time occurs in stations 3, 4, and 6, the other stations have idle times of 3, 3, 2, and 7 time units.

One of the optimal solutions to the SALBP-1 instance with $c = 11$ is ($\{1,3\}$, $\{2,4\}$, $\{5,6\}$, $\{7,8\}$, $\{9,10\}$) with the minimal number of $m^* = 5$ stations. An optimal SALBP-2 solution given $m = 6$ stations is ($\{3,4\}, \{1,5\}, \{2,7\}, \{6,8\}, \{9\}, \{10\}$) with minimal cycle time $c^* = 10$. Given a SALBP-E instance with the number of stations restricted to 5 to 7 stations, the optimal line efficiency $E^* = 48/(5 \cdot 11) = 0.87$ is achieved by the SALBP-1 solution given above.

The versions of SALBP may be complemented by a secondary objective which consists of *smoothing station loads*, i.e., equalizing the station times (*vertical balancing*, cf. Merengo et al., 1999; *equal piles problem*, cf. Rekiek et al., 1999). One may minimize the smoothness index $SX = \sqrt{\sum_{k=1}^{m}(c - t(S_k))^2}$ provided that the combination (m, c) is optimal with respect to line efficiency (see, e.g., Moodie and Young, 1965; Rachamadugu and Talbot, 1991).

In our example with given combination $(m, c) = (5,11)$, the smoothness index favors the solution ($\{3,4\}, \{1,5\}, \{2,7\}, \{6,8\}, \{9,10\}$) with $SX = 4.36$ versus the other optimal SALBP-1 solutions (including the one with $SX = 5.39$ specified for $c = 11$ above), because the stations of the first solution are more equally loaded.

Since SALBP-F is an NP-complete feasibility problem, the optimization versions of SALBP, that may be solved by iteratively examining several instances of SALBP-F, are NP-hard (cf. Wee and Magazine, 1982; Scholl, 1999, Chapter 2.2.1.5). Despite the principal NP-hardness of the problem class, there are considerable differences concerning the difficulty of solving single instances. In order to distinguish different classes with respect to the complexity of instances, different measures like the *order strength* or the *task time variability ratio* have been developed (for surveys see Scholl, 1999, Chapter 2.2.1.5; Driscoll and Thilakawardana, 2001).

Due to the complexity of SALBP, formulating a mathematical model and solving it by standard optimization software is no realistic choice for finding an optimal solution in case of real-world instances of ALBP (cf. Section 3.6). Thus, we do without describing such models but refer to Scholl (1999, Chapter 2.2.1.2) where different models are presented. Moreover, see Ugurdag et al. (1997), Pinnoi and Wilhelm (1997a), Bockmayr and Pisaruk (2001), and Peeters and Degraeve (this issue).

In what follows, we describe solution procedures for the different versions of SALBP using the notations of Table 2. SALBP-1 is one of the classical optimization problems that has been studied intensively since almost 50 years. The most relevant developments are effective branch-and-bound procedures, including intelligent branching schemes and a variety of bounding procedures (Section 3), flexible priority rule based procedures and modern meta-heuristics like tabu search and genetic algorithms and their problem-specific application (Section 5). Solution procedures for SALBP-2 and SALBP-E are mostly search methods based on repeatedly solving SALBP-F instances by SALBP-1 procedures (Section 4), others are directly applied heuristics similar to those for SALBP-1 (Section 5). Modifying assumptions of SALBP leads to generalized problems, solution procedures for which are usually based on such for SALBP (cf. Becker and Scholl, this issue).

Table 2
Notations

n	Number of tasks; index $j = 1,\ldots,n$
V	Set of tasks; $V = \{1,\ldots,n\}$
m	Number of stations; index $k = 1,\ldots,m$
m^*	Optimal number of stations
LM, UM	Lower, upper bound on m
c, c^*	Cycle time, optimal cycle time
LC, UC	Lower, upper bound on c
t_j	Task time of task $j = 1,\ldots,n$
t_{min}, t_{max}, t_{sum}	Minimal, maximal, total task time
p_j	Station requirement of task j; $p_j = t_j/c$
P_j (P_j^*)	Set of direct (all) predecessors of task j
F_j (F_j^*)	Set of direct (all) followers of task j
S_k, $t(S_k)$	Station load, station time of station k; $t(S_k) = \sum_{j \in S_k} t_j$, $k = 1,\ldots,m$
I_k	Idle time of station k; $I_k = c - t(S_k)$
a_j, n_j	Head, tail of task j
E_j, L_j	Earliest, latest station for task j

Former surveys covering SALBP procedures are given by Buxey et al. (1973), Baybars (1986a), Shtub and Dar-El (1989), Ghosh and Gagnon (1989), Erel and Sarin (1998), Scholl (1999, Chapter 1) as well as Rekiek et al. (2002b).

3. Exact solution procedures for SALBP-1

Many operations researchers have been engaged with developing effective solution procedures for exactly solving SALBP-1. This has resulted in about two dozens of procedures which can be subdivided into branch and bound (B&B) procedures and dynamic programming (DP) approaches. In the following, we summarize the most effective key developments without giving closed statements of single procedures. Further (procedure oriented) surveys are given by Baybars (1986a), Ghosh and Gagnon (1989), and Scholl (1999, Chapter 4.1).

Due to their importance for solving SALBP-1 exactly, we begin with methods for computing lower bounds on the number of stations (though they are usually not used in DP approaches) before different enumeration schemes and possibilities for reducing the effort of enumeration are discussed.

3.1. Lower bounds

SALBP-1 is to minimize the number m of stations given the cycle time c. Due to the integrality of m, only a relatively small number of objective values are possible. Therefore, it can be expected that a large number of line balances is feasible for each feasible value of m. That is, finding a line balance given the optimal number m^* of stations (solving the corresponding SALBP-F instance) is often easier than proving a certain number of stations $m < m^*$ to be infeasible. As a consequence, knowing sharp *lower bounds* LM on m^* is helpful in solving SALBP-1 instances provided that the bounds can be computed efficiently.

We describe and categorize some of the most important bound arguments for SALBP-1 (a more comprehensive survey is given by Scholl, 1999, Chapter 2.2.2.1, and Sprecher, 1999).

3.1.1. Bin packing bounds

SALBP-1 reduces to the bin packing problem BPP-1 (distributing a number of items to a minimum number of fixed-capacity bins; cf. Martello and Toth, 1990) by ignoring the precedence relations. That is, BPP-1 is a relaxation of SALBP-1 and lower bounds for BPP-1 are valid lower bounds for SALBP-1, too. We specify only some of those bounds (further ones can be found in Martello and Toth, 1990; Scholl et al., 1997; Fekete and Schepers, 2001; Alvim et al., 2003).

Total capacity bound (Baybars, 1986a). The most obvious bound LM1 follows from the inequality $m \cdot c \geqslant t_{sum}$, i.e., the total time (total capacity) available on the line must be no smaller than the total work content:

$$LM1 := \lceil t_{sum}/c \rceil = \left\lceil \sum_{j=1}^{n} p_j \right\rceil. \quad (1)$$

Simple counting bounds (Johnson, 1988). A bound LM2 is obtained by counting the number of tasks j with $t_j > c/2$ (i.e., $p_j > 1/2$), because all of those tasks have to be assigned to different stations. LM2 can be strengthened by adding half of the number of tasks (rounded up to the next integer if necessary) with task time $c/2$, because two of them may share one station.

A further bound LM3 generalizes LM2 with respect to thirds of the cycle time and is computed by adding up weights which are determined as follows: All tasks j with $p_j > 2/3$ are given the weight 1, because they cannot be combined with any other of the tasks considered. Tasks with $p_j \in (1/3, 2/3)$ get the weight 1/2, because two of them may share a station. The tasks with $p_j = 1/3$ and $p_j = 2/3$ are weighted with 1/3 and 2/3, respectively.

Extended bin packing bounds. The logic behind the simple counting bounds is combined and extended in several manners in order to define more sophisticated bound arguments (cf. Martello and Toth, 1990; Labbé et al., 1991; Berger et al., 1992; Scholl et al., 1997).

3.1.2. One-machine scheduling bound

The one-machine scheduling bound LM4 (Johnson, 1988) relies on relaxing SALBP-1 to a *one-machine scheduling problem*. Tasks are interpreted

as jobs $j = 1,\ldots,n$ with "processing times" $p_j = t_j/c$ which have to be successively performed on a single machine. After processing job j, a certain amount of time (called *tail* n_j) is necessary before the job is terminated. The objective of the one-machine problem is to find a sequence of all jobs for which the *makespan*, i.e., the time interval from the start of the first to the termination of the last job, is minimized. An optimal solution for this problem is achieved by processing the jobs in order of non-increasing tails. Let $\langle h_1,\ldots,h_n]$ be such a job ordering and tails n_j been given, the minimum makespan is:

$$\mathrm{MS} = \max\{p_{h_1} + n_{h_1}, p_{h_1} + p_{h_2} + n_{h_2}, \ldots, p_{h_1} + \cdots + p_{h_n} + n_{h_n}\}. \quad (2)$$

In the case of SALBP-1, a tail n_j of a task j is a lower bound on the number of stations (not necessarily integral) needed by its successors in F_j^*. Thus, any bound argument for SALBP-1 can be applied to the subproblem defined by F_j^* for computing the tails n_j (without rounding off the bound values). The computation requires a fictitious source node $j = 0$ added to the precedence graph (with $t_0 = 0$ and directed arcs to all original source nodes). The tails are recursively computed in the reverse topological task numbering $j = n,\ldots,0$. Johnson (1988) proposes to apply LM1, LM2, and LM3 as well as (2) to the subproblem with tasks F_j^* (due to the reverse computation the tails of those tasks are already known). The largest of those (unrounded) bound values defines the tail n_j of a task j considered. Whenever the conditions $n_j < \lceil n_j \rceil$ and $p_j + n_j > \lceil n_j \rceil$ are fulfilled, task j cannot share the first of the $\lceil n_j \rceil$ stations required by its followers such that n_j can be rounded up to $\lceil n_j \rceil$. This rounding process is the core logic of LM4 responsible for the quality of the computed value. The final value $Z_1 = \lceil n_0 \rceil$ defines a first lower bound for SALBP-1.

By applying the same logic in a forward oriented manner (using heads instead of tails), a second bound $Z_2 = \lceil a_{n+1} \rceil$ may be computed for a fictitious sink node $n+1$ (cf. Scholl, 1999, p. 47). A *head* a_j is a lower bound on the number of stations required by all predecessors of task j and is computed analogously to tails. Combining the information given by heads and tails provides the overall bound $\mathrm{LM4} = \max\{\lceil a_j + p_j + n_j \rceil \mid j = 0, \ldots, n+1\}$ which covers Z_1 and Z_2 for $j = 0$ and $j = n+1$, respectively.

3.1.3. Destructive improvement bounds

The bound arguments described so far may be classified as being constructive, because they find (construct) optimal solutions to relaxed problems. A further class of bounds can be defined as being destructive because they try to contradict trial objective values (numbers of stations) in order to successively improve on an initial lower bound. Therefore, this general approach for computing lower bounds by a successive process of contradicting potential bound values has been called *destructive improvement* by Klein and Scholl (1999).

Earliest and latest stations bound LM5 (Saltzman and Baybars, 1987; Scholl, 1999, p. 48). By means of heads and tails (determined as described for LM4), we may compute earliest and latest stations to which a task j can be assigned in order not to exceed a certain number $\underline{m}$ of stations:

$$E_j = \lceil a_j + p_j \rceil \quad \text{and} \quad L_j(\underline{m}) = \underline{m} + 1 - \lceil p_j + n_j \rceil \quad \text{for } j = 1,\ldots,n. \quad (3)$$

The destructive improvement concept for computing a bound LM5 works as follows: We start with a valid lower bound $\underline{m}$ computed by any constructive method. If for any task j the relation $E_j > L_j(\underline{m})$ holds, j is not assignable to any of the $\underline{m}$ stations and $\underline{m}$ is increased by 1. This is repeated until $E_j \leqslant L_j(\underline{m})$ is true for all tasks. The current value of $\underline{m}$ defines the bound LM5.

SALBP-2 based bound LM6 (Scholl and Klein, 1997). Due to the "duality" of SALBP-1 and SALBP-2 (cf. Section 2), one may use lower bounds for SALBP-2 in order to compute bounds for SALBP-1 in the destructive improvement framework: For each trial value $\underline{m}$, a lower bound $c(\underline{m})$ on the cycle time is computed by a bound argument for SALBP-2 (see Section 4.1). If $c(\underline{m})$ is larger than the given cycle time c, $\underline{m}$ is increased by 1, and the process is repeated. Otherwise, the current value of $\underline{m}$ provides a lower bound LM6 on the number of stations for SALBP-1.

Table 3
Heads and tails of tasks

j	0	1	2	3	4	5	6	7	8	9	10	11
p_j	0	0.6	0.6	0.5	0.5	0.4	0.5	0.4	0.2	0.9	0.2	0
a_j	0	0	1	0	0.5	1.6	2	1.6	3.5	4	5	5.2
n_j	5.7	4.1	3	4	3.4	3	2.2	2.2	2	1	0	0

Interval bound LM7. A further destructive bound is proposed by Fleszar and Hindi (2003) who define station intervals $[m_1, m_2]$ with $1 \leqslant m_1 \leqslant m_2 \leqslant \underline{m}$ for a current lower bound value m. If a solution with m stations exists, the tasks in the set $\{j \in V | E_j \geqslant m_1 \text{ and } L_j(\underline{m}) \leqslant m_2\}$ must be assigned to at most $m_2 - m_1 + 1$ stations. If applying any SALBP-1 bound to this subproblem reveals that this is not possible for any interval $[m_1, m_2]$, $\underline{m}$ is contradicted as a feasible number of stations and is increased by 1. The process is repeated until no "destruction" of the current $\underline{m}$ is possible which then defines LM7.

Bin packing applications of destructive improvement are also applicable to SALBP-1 as mentioned earlier (cf. Scholl et al., 1997; Alvim et al., 2003).

Computational experiments show that among the bound arguments presented so far LM4 is the most powerful one (cf. Scholl, 1999, p. 242). However, due to the diversity of problem instances to be solved applying all arguments available is recommendable.

A further class of bounds is based on solving relaxations of mathematical model formulations for SALBP-1 by means of standard optimization packages. Peeters and Degraeve (this issue) present a Dantzig-Wolfe type formulation of SALBP-1 the LP-relaxation of which is solved using column generation combined with subgradient optimization. Computational experiments show that the resulting bound values are close to optimality. However, this must be paid by a high computational effort which is often higher than for exactly solving the unrelaxed problem.

3.1.4. Example for bound computation

For illustration purposes, we reconsider our example of Fig. 1 which is characterized by $t_{sum} = 48$, $t_{max} = 9$, and $t_{min} = 2$ as well as $c = 10$. The following bound values are obtained:

- LM1 $= \lceil 48/10 \rceil = 5$.
- LM2 $= 3 + \lceil 3/2 \rceil = 5$, because there are three tasks $(1,2,9)$ exceeding half of the cycle time and three further tasks $(3,4,6)$ having task time $c/2$. LM3 $= \lceil 1 \cdot 1 + 1/2 \cdot 7 \rceil = 5$ due to $p_9 > 2/3$ and $p_j \in (1/3, 2/3)$ for $j \in V - \{8,9,10\}$.
- We do without explaining the computation of LM4 in detail but give the heads and tails in Table 3. We get LM4 $= \lceil 5.9 \rceil = 6$ (defined by task 9). The rounding logic of LM4 becomes obvious considering, e.g., task 2 whose head a_2 is set to 1 because it is not possible to assign 2 to the same station as its direct predecessor 1 though the station requirement of task 1 is only $p_1 = 0.6$.
- LM5 is computed starting with, e.g., $\underline{m} =$ LM1 $= 5$. Table 4 contains the values for E_j and $L_j(5)$. Due to $E_j > L_j(5)$ for $j = 8,9,10$, these tasks cannot be assigned to any of the 5 stations, and $\underline{m}$ is increased to 6. Since the condition $L_j(6) \geqslant E_j$ holds for all tasks j ($L_j(6) = L_j(5) + 1$; cf. Table 4), LM5 gets the value 6. Beyond the bound computation, we get the information that tasks 8, 9, and 10 must be assigned to station 4, 5, and 6, respectively, in order to find a solution with no more than 6 stations.

Table 4
Earliest and latest stations

j	1	2	3	4	5	6	7	8	9	10
E_j	1	2	1	1	2	3	2	4	5	6
$L_j(5)$	1	2	1	2	2	3	3	3	4	5
$L_j(6)$	2	3	2	3	3	4	4	4	5	6

- For LM6 we may also start with $\underline{m} = 5$. A simple bound for SALBP-2 consists of considering the $\underline{m} + 1$ largest tasks. Provided that they are sorted in non-increasing order of task times, $c(\underline{m}) = t_{\underline{m}} + t_{\underline{m}+1}$ is a lower bound on the cycle time given $\underline{m}$ stations. In our example, we get $c(5) = 5 + 5 = 10 = c$ and, hence, LM6 = 5.
- Starting the computation of LM7 with $\underline{m} = 5$, one finds a contradiction considering the subproblem with station interval $[1,2]$ and the task set $\{1,\ldots,5\}$ due to $\mathrm{LM1} = \lceil 26/10 \rceil = 3 > 2$. No further contradiction arises for $\underline{m} = 6$ such that LM7 = 6 is determined.

3.2. Construction schemes

Most exact algorithms are based on enumerating feasible solutions by successively assigning tasks or subsets of tasks to stations. Therefore, these algorithms consider *partial solutions* containing a number of already assigned tasks and (partial) station loads S_k $(k = 1,2,\ldots)$, while the remaining tasks and station idle times constitute a *residual problem*.

For explaining solution procedures we need some further terms and definitions:

- A yet unassigned task j is *available* if all preceding tasks $h \in P_j^*$ have already been assigned to a station.
- An available task is *assignable* to a station k if all preceding tasks have already been assigned to station k or an earlier station $1,\ldots,k-1$ and the current idle time is sufficient.
- A station load S_k is *maximal* if no available task is assignable to station k.
- A subset of tasks $S \subseteq V$ is *feasible* if S also contains the predecessors of each task included.

Almost every solution procedure for SALBP-1 is based on either of the two following *construction schemes*, which define the principle way of assigning tasks to stations:

- *Station-oriented assignment*. In any step of a station-oriented procedure a complete load of assignable tasks is built for a station k, before the next one $k + 1$ is considered.
- *Task-oriented assignment*. Procedures which are task-oriented iteratively select a single available task and assign it to a station, to which it is assignable.

3.3. Dominance rules

Most procedures use *dominance* rules to reduce the enumeration effort. Such a rule compares partial solutions (and corresponding residual problems) in order to find dominance relationships which allow for excluding (dominated) partial solutions without explicitly completing them to full solutions. The concept of dominance is based on the fact that all partial solutions which cannot be completed to a solution with minimal number of stations and even all of those but one which can be completed to an optimal solution could be eliminated from further consideration. Since it is usually not known in advance into which category the partial solutions fall, a dominance relationship between two partial solutions P_1 and P_2 is established whenever it can be proved that the optimal completion of P_1 does not require more stations than that of P_2. Whenever such a dominance relationship is found, P_2 can be excluded from further consideration.

Due to their importance in developing effective procedures for SALBP-1 we explain some of the most powerful dominance rules (furthermore, see Scholl, 1999, Chapter 4.1, and Sprecher, 1999).

• *Maximum load rule* (Jackson, 1956). It excludes each partial solution P_2 which contains one or more completed but *non-maximal station loads*, because there exists at least one another partial solution P_1 with the same number of maximally loaded stations, where additional tasks are assigned. In such case the completion of P_1 does not require more stations than that of P_2. The maximum load rule does not require explicit comparisons between several partial solutions, because it is sufficient to examine the maximality of station loads while these are constructed (Johnson, 1988).

Example. Given a cycle time of 10 and the precedence graph of Fig. 1, $S_1 = \{1\}$ and $S_1 = \{3,4\}$ are maximal loads for station 1, while $S_1 = \{3\}$ is non-maximal. Provided that $S_1 = \{3,4\}$ has already been built, $S_2 = \{1,5\}$ is the only maximal load available for station 2.

• *Jackson dominance rule* (Jackson, 1956, strengthened by Scholl and Klein, 1997). This rule 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.

A task h *potentially dominates* a task j that is not related to h by precedence, if $F_j \subseteq F_h^*$ and $t_j \leqslant t_h$ hold. If $t_j = t_h$ and $F_j = F_h$, the lower-numbered task is defined to be dominating.

The Jackson rule excludes a maximal station load S_k from consideration if at least one task $j \in S_k$ can be replaced by an available not yet assigned task h which potentially dominates j, and the resulting station time $t(S_k) - t_j + t_h$ does not exceed the cycle time c.

This rule utilizes the fact that all successors of task j are successors of h as well and cannot start before h is finished. Hence, the sequence of j and h is not important for the successors of j such that replacing h by j does not exclude later task assignments which would be possible when j remained. The condition $t_j \leqslant t_h$ guarantees that the station utilization will not decrease if h replaces j in S_k.

Example. In Fig. 1, the following pairs of potential dominance (h,j) are present: $(1,4)$, $(2,6)$, $(3,7)$, $(4,7)$, $(5,7)$, $(6,7)$. This results, e.g., in the partial solution $P_1 = (\{1\},\{3,4\},\{2,5\})$ dominating $P_2 = (\{1\},\{3,4\},\{2,7\})$ and also $P_3 = (\{1\},\{3,4\},\{5,6\})$.

• *Feasible set dominance rule* (Schrage and Baker, 1978; Nourie and Venta, 1991; Scholl and Klein, 1999). It extends the logic of the maximal load rule to *feasible subsets of tasks*. Provided that a partial solution P_2 is constructed in forward direction (from early to late stations) and the maximum load rule is applied to all m_2 stations already loaded, the corresponding feasible subset T_2 of tasks is assigned to these m_2 stations. Whenever another partial solution P_1 has already been considered which assigns a feasible task set T_1 (with $T_2 \subseteq T_1$) to $m_1 \leqslant m_2$ stations, P_2 is dominated by P_1 and excluded.

In order to apply the feasible set dominance rule, it is necessary to store all feasible subsets of tasks already enumerated together with their minimum station requirements in an efficient manner with respect to storage space and retrieval speed. For this purpose several approaches are available: Held et al. (1963) propose a recursive addressing algorithm which assigns a unique address to each feasible subset using the address space compactly. Because this addressing algorithm is very time-consuming, Schrage and Baker (1978) develop a simpler procedure based on task labels which have to be summed up to quickly compute the address of a subset. However, this addressing is not compact and therefore requires a lot of storage space. In order to overcome this disadvantage, Nourie and Venta (1991) define a tree structure for efficiently storing all feasible subsets which outperforms the previous approaches. A similar approach is proposed by Sprecher (1999). All those methods are restricted to examining the dominance relation for the special case $T_2 = T_1$, the more general case $T_2 \subseteq T_1$ is examined in a solution procedure for the related resource constrained scheduling problem (cf. Klein and Scholl, 2000).

Example. The partial solution $P_1 = (\{3,4\},\{1,5\},\{2,7\})$ assigns the feasible set $T_1 = \{1,\ldots,5,7\}$ to $m_1 = 3$ stations. Another partial solution $P_2 = (\{1\},\{3,4\},\{2,5\})$ corresponds to $T_2 = \{1,\ldots,5\}$ and requires $m_2 = 3$ stations, too. Thus, P_2 is dominated by P_1.

• *Station ordering rules*. Because finding a single optimal solution is sufficient, it is not necessary to consider different partial solutions which have the same station loads but differ only in the sequence these loads are assigned to stations. Several rules which attempt to find respective dominance relationships in an efficient manner are proposed by Johnson (1988), Scholl and Klein (1997) as well as Sprecher (1999).

Example. The partial solutions $P_1 = (\{1\},\{2,7\},\{3,4\})$ and $P_2 = (\{1\},\{3,4\},\{2,7\})$ are identical except for the sequence of the second and third load. Thus, one can be excluded.

Computational experience reveals that applying (computationally inexpensive) dominance rules is very important for the computation times of exact procedures (cf. Johnson, 1988; Nourie and Venta, 1991, 1996; Scholl and Klein, 1997; Scholl, 1999, p. 242).

3.4. Reduction rules

Besides dominance rules several procedures use *reduction rules* to lower the computational effort for solving SALBP-1. Such rules try to modify problem data, i.e., task times or precedence relations, such that the number of solution alternatives can be reduced or bounds may get improved values. However, the reduced problem must have at least one optimal solution in common with the original problem.

- *Task time incrementing rule* (Johnson, 1988; Scholl, 1999, p. 117). The task times of all tasks which cannot share a station with any other task are set to c. A simple sufficient condition for incrementing the time t_j of a task j to c is $t_j + t_{\min} > c$. Improved conditions are given by Talbot and Patterson (1984), Sprecher (1999), and Fleszar and Hindi (2003).

Example. In the instance given by Fig. 1 and $c = 10$, we find that the time of task 9 can be increased to 10. Inspecting the precedence structure reveals that the time of task 10 can be set to 10, too. Doing so, immediately leads to the improved $LM1 = \lceil 57/10 \rceil = 6$.

- *Prefixing* (Scholl and Klein, 1999). Whenever it becomes clear that a task can only be assigned to one specific station in order to find an improved solution (or one with a predefined number of stations) this task can be assigned (prefixed) to the respective station. Provided that UM denotes a valid upper bound on the number of stations, $E_j = L_j(\mathrm{UM} - 1)$ is a sufficient condition which allows for prefixing task j to station E_j because an improved solution must not have more then $\mathrm{UM} - 1$ stations.

Example. Let us assume that a feasible solution with $\mathrm{UM} = 7$ stations is already known. Then the tasks 8, 9, and 10 can be prefixed to the stations 4, 5, and 6, respectively (cf. Table 4).

- *Additional precedence relations* (Fleszar and Hindi, 2003). Consider two tasks h and j which are not related by precedence. If $m(j,h) = \lceil a_j + p_j + p_h + n_h \rceil > \mathrm{UM} - 1$, an additional arc (h,j) is added to the precedence graph, because performing j before h cannot lead to an improved solution. If $m(h,j) > \mathrm{UM} - 1$ is additionally true, no solution with less than UM stations exists. That is, examining precedence relations in both directions is a means that may be applied within destructive improvement bounds (cf. Klein and Scholl, 1999).

Example. Again starting with $\mathrm{UM} = 7$, we can add the arc $(3,2)$ due to $m(2,3) = \lceil 6.1 \rceil > 6$ and $m(3,2) = \lceil 4.1 \rceil < 6$ (cf. Table 3). Trying to contradict $\underline{m} = 5$ or, equivalently, trying to improve on $\mathrm{UM} = 6$, a contradiction is found considering the tasks 1 and 3 because of $m(1,3) > 5$ and $m(3,1) > 5$.

- *Task conjoining rule* (Fleszar and Hindi, 2003). If all (potential) maximal loads containing a particular task j also contain another task h, both nodes can be merged into one node thereby uniting the sets of predecessors and successors.

Example. Provided that the arc $(3,2)$ has been added as described in the rule above, the only maximal load containing task 3 is $\{3,4\}$ such that both tasks can be merged.

Fleszar and Hindi (2003) define further rules which may allow for decomposing the precedence graph and improving on lower bounds by successively adding dummy tasks and increasing task times. Their computational experiments show that reduction rules have a great potential for improving solutions found by any heuristic if they are applied prior to the heuristic. Furthermore, they allow for computing sharper lower bounds.

3.5. Dynamic programming procedures

As already mentioned, exact solution procedures for SALBP-1 are based either on dynamic programming or branch and bound.

The first *dynamic programming* (DP) procedure developed by Jackson (1956) and modified by Held et al. (1963) subdivides the solution process in stages which correspond to stations, i.e., it follows

the station-oriented construction scheme. States are given by the feasible subsets of tasks already assigned at a given stage. The optimal solution is searched for stage-by-stage in a forward recursion, i.e., the procedure enumerates all first station loads before it considers all additional second station loads and so on (breadth-first search). In order to restrict the enumeration effort, only maximal station loads not being dominated by the Jackson dominance rule (cf. Section 3.3) are built.

If in Jackson's approach the states are represented by nodes and the station loads, by arcs which are weighted with the corresponding station idle times, SALBP-1 is transformed to an equivalent *shortest path problem*. Each path in this graph corresponds to a feasible solution and each shortest path to an optimal solution of SALBP-1 (cf. Gutjahr and Nemhauser, 1964). Since the number of nodes grows exponentially with the number of tasks, it is generally not possible to construct the complete graph. Therefore, Easton et al. (1989) incorporate lower and upper bounds as well as dominance rules to reduce the size of the graph. Furthermore, their procedure is designed to be applied in forward and backward direction, respectively.

Further DP procedures apply task-oriented construction schemes. The procedures of Held et al. (1963) and Schrage and Baker (1978) perform a forward recursion and store states together with their minimum station requirement in a memory using different addressing schemes (see feasible set dominance rule; cf. Section 3.3). However, both procedures have large memory requirements because they do not strictly work stage-by-stage. The procedures of Lawler (1979) and Kao and Queyranne (1982) improve on this drawback and get by with considerably less memory.

3.6. Branch and bound procedures

In the last decades, a considerable number of *branch and bound* (*B&B*) approaches have been proposed in the literature. Table 6 gives a survey of these procedures and characterizes them briefly by means of the notation in Table 5.

Table 5
Possible components

LMi	Lower bound LMi ($i = i,\ldots,6$)
ML	Maximum load rule
FS	Feasible set dominance rule
JD	Jackson dominance rule
SO	Station ordering dom. rule
TI	Task time incrementing rule
PF	Prefixing
RN	Task renumbering
HS	Additional heuristic
FW/BW	Constructing solutions in forward/backward direction

3.6.1. Search strategies

Besides the construction scheme (station- or task-oriented), the B&B procedures differ with respect to search strategy.

- In a *depth-first search* (DFS), a single branch of the tree is developed until a leaf node is reached or the current node is fathomed. On its way back to the root, the search follows the first possible alternative branch, i.e., each node is completely developed before its ancestor nodes are revisited. Most commonly, DFS is organized as *laser search* (LS), i.e., in each node of the current branch, only one descending node is built and developed at a time. Alternatively, all descending nodes may be generated and sorted by a priority rule (e.g., according to non-decreasing lower bound values). The one with the highest priority is branched first. The remaining nodes are stored in a *candidate list* and are developed according to the priority order at each revisit of the node. This derivative is called DFS with *complete node development* (DFSC).
- A *minimal lower bound strategy* (MLB) always chooses a not yet developed node which has the minimum value of a lower bound (see Section 3.1) from a candidate list. This node is completely branched by constructing all descending nodes. The latter are stored in the list which is sorted according to non-decreasing bound values, while the current node is dropped. At the beginning, the list only contains the root node.

Irrespective of the strategy used, a node is fathomed when its (*local*) *lower bound* (computed explicitly or inherited from the father node) is not smaller than the global upper bound UM

(number of stations in the currently best known solution), because it does not have an optimal solution with less than UM stations.

While DFS considers (and stores) only a small part of the enumeration tree at a time, MLB has to manage a large number of not yet developed nodes considerably restricting its applicability to SALBP-1 and other combinatorial optimization problems. Therefore, the most effective algorithms for SALBP-1 presented recently are based on some type of DFS strategy.

However, both extreme versions of DFS discussed above have potential disadvantages, too: LS considers the different subproblems in an (arbitrary) sequence induced by the task labeling taking the risk to examine large subtrees before promising parts of the enumeration tree are entered and improved solutions (with decreased UM) are found. That is, LS tends to spend a lot of time in investigating many solutions which could have been fathomed if an improved UM value was determined earlier. In order to reduce this negative effect, most procedures using LS *renumber* the tasks concerning some priority rule such that the first subproblems generated contain promising task assignments which have the potential of being part of an improved solution.

DFSC explicitly tries to avoid the drawback of LS by generating all subproblems of a current node before they are examined in order of non-decreasing lower bound values. So, promising partial solutions are considered first, but all subproblems have to be generated irrespective of the difficulty in finding an improved UM value. That is, even in case of finding the optimal solution to a node in the first branch followed, all subproblems have been generated before.

The so-called *local lower bound method* (LLBM) aims at finding a reasonable compromise between these extreme positions (cf. Scholl and Klein, 1997). The subproblems of a current node are subdivided into two classes: class I contains all sub-

Table 6
Survey of branch and bound procedures for SALBP-1

Station-oriented	Task-oriented
DFS	
• Mertens (1967): FW, LS, LM1, RN • Johnson (1981): FW, DFSC, LM1, modified LM2,3, ML • Betts and Mahmoud (1989): similar to Johnson (1981), enumeration based on Hoffmann (1963) • Berger et al. (1992): similar to Hackman et al. (1989) restricted to out-tree type precedence graphs, LM1 and extended bin packing bound • EUREKA, Hoffmann (1992, 1993): FW-BW (successively), LS, LM1; lower bound method: repeated application for increasing trial numbers of stations; application of heuristic of Hoffmann (1963) if no feasible solution is found	• Talbot and Patterson (1984), based on additive algorithm of Balas (1965): FW, LS, reduction tests based on LM1, TI • Saltzman and Baybars (1987): parallel FW-BW, LS, RN, LM1,5, LM2,3 modified like Johnson (1981), TI, HS • FABLE, Johnson (1988, 1993): FW, LS, RN, LM1-4, ML, JD, FS (restricted), TI, SO • Nourie and Venta (1991, 1996): branching like FABLE, FW, LS, RN, LM1, ML, FS, HS • Sprecher (1999): branching like FABLE, FW-BW (static direction switching), LS, RN, LM1-6, ML, JD, FS, TI, SO • Sprecher (2003): distributed version of Sprecher (1999) for parallel computers
MLB	
• Jaeschke (1964): FW, LM1, ML • Van Assche and Herroelen (1979): FW, LM1, ML, FS (restricted) • Hackman et al. (1989): FW, LM1, ML, FS (restricted), HS	
LLBM	
• SALOME-1, Scholl and Klein (1997, 1999): FW-BW (bidirectional), RN, LM1-6, ML, JD, FS (Nourie and Venta, 1991), PF, TI, SO • Bock and Rosenberg (1998), Bock (2000): distributed version of SALOME-1	

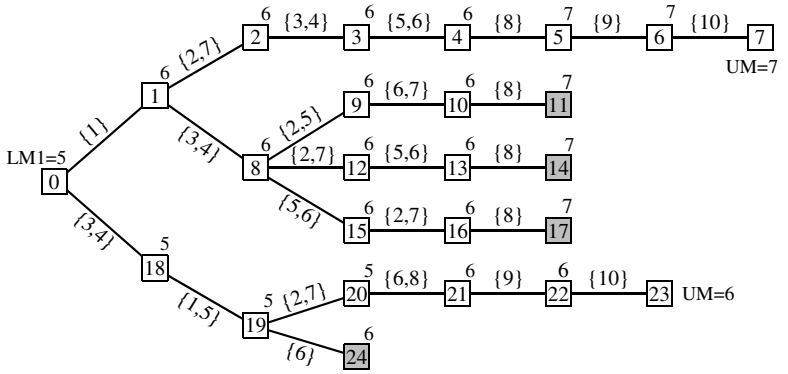

Fig. 2. Enumeration tree with laser search and bound LM1.

problems which have the same local lower bound value LLB as the current node, class II contains all remaining subproblems which have an increased LLB. Following the basic principle of DFSC, the subproblems of the first class are branched before those of the second class. This is implemented as follows: the subproblems are generated following a task renumbering as in case of LS. Each subproblem is judged concerning its membership to class I or II. Class II-problems are dropped and re-enumerated later if necessary, class I-problems are chosen for branching. After having examined the corresponding subtrees, the following situations may occur in the current node:

(1) If an improved upper bound UM = LLB has been found, the current node can be fathomed immediately without generating further subproblems.
(2) If an improved upper bound UM = LLB + 1 has been found, all class II-problems can be dropped, i.e., the search stops after having examined all class I-problems.
(3) If all class I-problems have been examined, LLB is increased by 1 (the current node has no solution with at most LLB stations) starting the examination of the class II-problems if LLB remains smaller than UM. Otherwise, the current node is fathomed.

3.6.2. Comparison of search strategies

We compare the different search strategies by means of our example defined by $c = 10$ and the precedence graph of Fig. 1. In order to concentrate on these strategies, we only apply the simple lower bound LM1 and do without dominance and reduction rules except for the maximum load rule. As construction scheme, we use the station-oriented assignment. Fig. 2 shows the enumeration tree obtained by applying the LS strategy, where the node numbering indicates the sequence of node generation (the tree is developed from left to right and from top to bottom). The initial bound of the root node 0 is given by $LM1 = \lceil 48/10 \rceil = 5$, the local lower bounds of the other nodes are given as node weights. In the first branch, a feasible solution with UM = 7 is found. This upper bound allows for fathoming the shaded nodes 11, 14, and 17.

In node 23, an improved solution with UM = 6 stations is found, such that node 24 can be fathomed. Since all nodes with LM1 = 5 are now completely branched, the procedure stops with the optimal solution $\mathbf{S}^* = (\{3,4\},\{1,5\},\{2,7\},\{6,8\},\{9\},\{10\})$ with 6 stations. Solving this problem instance to optimality requires examining 25 nodes of the enumeration tree.

The DFSC strategy starts with completely branching node 0 thereby building (and storing) the nodes 1 and 18. Computing LM1 for both nodes indicates that the search should continue with branching node 18. This generates only node 19 whose branching results in the nodes 20 and 24 with lower bound values 5 and 6, respectively. Therefore, the search continues with node 20 finally leading to the optimal solution with 6 stations. Backtracking leads to fathoming the nodes

24 and 1 and the procedure stops. In total, DFSC has to consider 9 nodes.

The LLBM only builds the branch leading to the optimal solution (nodes 0 and 18–23) without storing the nodes 1 and 24. After having found the solution with UM = 6, no further branching is required because the loads {1} in node 0 and {6} in node 19 would lead to class II-subproblems. So, LLBM requires the minimal number of 7 nodes.

The MLB strategy might produce the same tree as DFSC. However, due to several nodes having the same bound value at a time it might perform much worse. For example, the following sequence of selecting nodes for further development or fathoming is possible which requires examining 23 nodes: 0, 18, 19, 20, 1, 2, 3, 4, 8, 9, 10, 12, 13, 15, 16, 21, 22, 23, 24, 5, 11, 14, 17. That is, the performance of the procedure depends on resolving such tie break situations.

Of course, one may find other problem instances where MLB or DFSC or even LS perform better than LLBM but experimental results indicate that LLBM is the most effective scheme in many situations (cf. Scholl and Klein, 1997).

When these basic strategies are enriched with additional bound arguments, dominance and reduction rules the differences between the search strategies are reduced. For example, the nodes 12 and 15 are avoided by the Jackson dominance rule, because task 7 is potentially dominated by task 5 and task 6 by task 2. Moreover, node 11 would be avoided by the feasible set dominance rule due to node 5 and node 24 due to node 15. Nevertheless, 17 nodes remain for LS.

3.6.3. Comparison of construction schemes and complete procedures

Comparing the different construction schemes (task- versus station-oriented assignment) reveals a methodical drawback of the task-oriented approach, because it successively fills stations with tasks without controlling their final idle times. This drawback becomes apparent most clearly when the task-oriented assignment is connected with LS (as is the case with all task-oriented procedures currently on hand; cf. Table 6), because each partial station load and the following subtree must be completed even in cases where the finished station load is recognized as being inferior due to much idle time. That is, the search can less strictly be directed to finding promising partial solutions soon as in case of other search strategies.

Similarly to station-oriented LS procedures (see above), additionally applying bound arguments and dominance rules partially removes this disadvantage. However, this comes along with additional expense in computation time and storage space. This tradeoff can best be seen comparing FABLE (Johnson, 1988) and EUREKA (Hoffmann, 1992), because the first performs a task-oriented LS with a lot of bounding, dominance and reduction rules, while the latter performs a station-oriented LS almost without such rules. Experimental tests show that both procedures which were the benchmark procedures in the early nineties get similar results when EUREKA is restricted to the forward direction (Johnson, 1993; Hoffmann, 1993; Scholl, 1999, p. 238) despite their very different approaches. The tests have been performed using benchmark data sets of Talbot et al. (1986), Hoffmann (1990, 1992), and Scholl (1993) that are downloadable from http://www.assembly-line-balancing.de and described in Scholl (1999, p. 234).

Some improvements are obtained by including the static backward planning of EUREKA (Scholl, 1999, p. 239) which indicates that some problem instances are easier solved in the reverse direction. Significant further improvements are due to the LLBM connected with the flexible bidirectional branching of SALOME-1, which attempts to find the preferable planning direction in any node of the tree by some type of priority rule (Scholl and Klein, 1997). Additional components like dynamic prefixing and dynamic task renumbering as well as the FS rule together with the storage scheme of Nourie and Venta (1991, 1996) considerably speed up SALOME-1 (Scholl and Klein, 1999).

The only (serial) procedure which seems to be competitive to SALOME-1 is that of Sprecher (1999) who applies many of the components also contained in SALOME-1 but performs a task-oriented LS. The potential disadvantage of such type of enumeration (see above) is compensated by

utilizing improved knowledge on characteristics of optimal solutions.

Significant speed ups are obtained by parallelized versions of SALOME-1 and the Sprecher algorithm (cf. Bock and Rosenberg, 1998; Bock, 2000; Sprecher, 2003). This is due to the fact that simultaneously searching in different parts of the solution space enlarges the probability of finding an optimum soon.

SALBP-1 can be formulated as a special case of the generalized resource-constrained project scheduling problem (GRCPSP) as shown in De Reyck and Herroelen (1997) and Sprecher (1999). However, applying exact procedures for GRCPSP is not competitive to specialized SALBP procedures (cf. De Reyck and Herroelen, 1997).

Recently, Bockmayr and Pisaruk (2001) developed a branch and cut procedure based on integer programming formulations with additional valid inequalities and constraint programming techniques. Pinnoi and Wilhelm (1997a,b, 1998) also propose branch and cut procedures for SALBP-1 connected with vertical balancing (cf. Section 2) as well as for more general problems.

Though these researchers invest a lot of work in finding sharp valid inequalities, their computational experiments clearly show that such procedures cannot compete with state-of-the-art enumeration based B&B algorithms. The same is true for DP procedures (cf. Section 3.5).

4. Exact solution procedures for SALBP-2 and SALBP-E

4.1. Lower bounds for SALBP-2

In what follows, we describe only a few arguments for computing lower bounds for SALBP-2, further ones are proposed by Scholl (1999, Chapter 2.2.3.1). As in case of SALBP-1, these bounds utilize relationships to other problems and the destructive improvement approach (Section 3.1).

Bound LC1 (McNaughton, 1959). By analogy with LM1, a simple lower bound on the cycle time follows from the necessary feasibility condition $m \cdot c \geqslant t_{\text{sum}}$. Furthermore, the indivisibility of tasks requires that $c \geqslant t_{\max}$. Hence, a lower bound for SALBP-2 is given by $\text{LC1} := \max\{t_{\max}, \lceil t_{\text{sum}}/m \rceil\}$.

Bound LC2 (Klein and Scholl, 1996). SALBP-2 passes into a parallel machine problem with the objective of minimizing the makespan by omitting the precedence relations. A lower bound for the parallel machine problem and, thus, for SALBP-2 is obtained as follows provided that the tasks are renumbered according to non-increasing task times, i.e., $t_j \geqslant t_{j+1}$ for $j = 1,\ldots,n$. Consider the $m+1$ largest tasks $1,\ldots,m+1$. A lower bound on the cycle time for this reduced problem is $t_m + t_{m+1}$, the sum of the two smallest task times, because at least one station contains two tasks. For the $2m+1$ largest tasks, a lower bound is given by the sum of the three smallest operation times, $t_{2m-1} + t_{2m} + t_{2m+1}$. In general, a lower bound LC2 is defined as

$$\text{LC2} := \max\left\{ \sum_{i=0}^{k} t_{k\cdot m+1-i} \mid k = 1,\ldots,\lfloor (n-1)/m \rfloor \right\} \tag{4}$$

Bound LC3 (Scholl, 1999, p. 56). This bound is based on earliest and latest stations and destructive improvement. Given an initial lower bound $\underline{c}$ on the cycle time, earliest and latest stations are obtained from heads and tails (depending on $\underline{c}$) as follows (cf. Section 3.1):

$$E_j(\underline{c}) := \lceil a_j(\underline{c}) + p_j(\underline{c}) \rceil \text{ and}$$
$$L_j(\underline{c}) := m + 1 - \lceil p_j(\underline{c}) + n_j(\underline{c}) \rceil \quad \text{for } j = 1,\ldots,n. \tag{5}$$

LC3 is computed by successively increasing $\underline{c}$ until $E_j(\underline{c}) \leqslant L_j(\underline{c})$ is obtained for all $j = 1,\ldots,n$.

Bound LC4 (Scholl, 1999, p. 57). By subdividing the interval $[1,m]$ of all stations into two subintervals $[1,i]$ and $[i+1,m]$, we obtain two aggregate "stations". For a valid lower bound $\underline{c}$, these "stations" have to observe the "cycle times" $i \cdot \underline{c}$ and $(m-i)\cdot \underline{c}$, respectively. This modified problem is solved as follows:

All tasks with $L_j(\underline{c}) \leqslant i$ are assigned to the first "station" and those with $E_j(\underline{c}) > i$ are assigned to the second one. The remaining tasks and the remaining idle times of the two aggregate "stations" constitute a residual problem. This problem

can be formulated as the feasibility version of the bin packing problem with two bins and capacities equal to the remaining idle times of the two aggregate "stations" (2Bin-F for short). The question of this feasibility problem reads as: "Does there exist a partition of the remaining tasks (items) into at most two bins with given capacities?" Even though 2Bin-F is NP-complete, it can often be solved in a reasonable computation time, because the residual problem is usually much smaller than the original SALBP-2 instance and only two bins are considered. The problem may be solved by any exact procedure for the bin packing problem (cf., e.g., Martello and Toth, 1990, Chapter 8; Scholl et al., 1997). If the remaining idle times of the two "stations" are different, equal-sized bins are obtained by pre-assigning a fictitious item, whose weight is equal to the idle time difference, to form a bin with smaller capacity as required.

Following the destructive improvement approach, LC4 is given by the minimal trial value c for which 2Bin-F has a positive outcome for all possible subdivisions of $[1,m]$ into two aggregate "stations" $[1,i]$ and $[i+1,m]$, i.e., for $i=1,\ldots,m-1$.

Example. We consider a SALBP-2 instance with the precedence graph of Fig. 3 and $m=5$. With

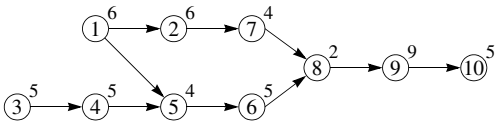

Fig. 3. Precedence graph.

$t_{sum}=51$ and $t_{max}=9$, we get $LC1=\max\{9,\lceil 51/5\rceil\}=11$. Because the six largest tasks are 9, 1, 2, 3, 4, and 6, $LC2=t_4+t_6=10$ is obtained.

Taking $\underline{c}=\max\{LC1,LC2\}=11$ as initial bound for computing LC3, we get the corresponding station requirements, heads and tails given in the first rows of Table 7. Due to $a_{11}(11)>5$, it is clear that $\underline{c}=11$ is no feasible cycle time and is increased to $\underline{c}=12$. The further rows of Table 7 contain the respective station requirements, heads and tails as well as earliest and latest stations. Obviously, the condition $E_j(12)\leqslant L_j(12)$ holds for $j=1,\ldots,10$ and $LC3=12$ cannot be "destroyed" as a potential cycle time.

For computing LC4, we start with the best known lower bound $\underline{c}=\max\{LC1,LC2,LC3\}=12$ and use the values $E_j(12)$ and $L_j(12)$ of Table 7.

For $\mathbf{i=1}$ we get two "stations" with "cycle times" $c_1=12$ and $c_2=48$. The tasks 5–10 are assigned to "station" 2 ($S'_2=\{5,\ldots,10\}$) and task 1 to "station" 1 ($S'_1=\{1\}$). The two bins have residual capacities $\kappa_1=6$ and $\kappa_2=19$, and the tasks 2, 3, and 4 remain. A feasible solution of 2Bin-F is given by the sets $S''_1=\{2\}$ and $S''_2=\{3,4\}$.

$\mathbf{i=2}$. $c_1=24$, $c_2=36$, $S'_1=\{1,3,4\}$, $S'_2=\{6,8,9,10\}$; $\kappa_1=8$, $\kappa_2=15$; $S''_1=\{2\}$, $S''_2=\{5,7\}$.
$\mathbf{i=3}$. $c_1=36$, $c_2=24$, $S'_1=\{1,\ldots,7\}$, $S'_2=\{8,9,10\}$; no residual problem.
$\mathbf{i=4}$. $c_1=48$, $c_2=12$, $S'_1=\{1,\ldots,9\}$, $S'_2=\{10\}$; no residual problem.

Since no infeasibility is found, $LC4=12$ cannot be disproved as a feasible cycle time.

Table 7
Heads and tails of tasks for $\underline{c}=11$ and $\underline{c}=12$

j	0	1	2	3	4	5	6	7	8	9	10	11
$p_j(11)$	0	$0.\overline{54}$	$0.\overline{54}$	$0.\overline{45}$	$0.\overline{45}$	$0.\overline{36}$	$0.\overline{45}$	$0.\overline{36}$	$0.\overline{18}$	$0.\overline{81}$	$0.\overline{45}$	0
$a_j(11)$	0	0	1	0	$0.\overline{45}$	$1.\overline{45}$	2	$1.\overline{54}$	$3.\overline{18}$	4	5	$5.\overline{45}$
$n_j(11)$	5	4	3	4	$3.\overline{36}$	3	$2.\overline{18}$	$2.\overline{18}$	2	1	0	0
$p_j(12)$	0	0.5	0.5	$0.41\overline{6}$	$0.41\overline{6}$	$0.\overline{33}$	$0.41\overline{6}$	$0.\overline{33}$	$0.1\overline{6}$	0.75	$0.41\overline{6}$	0
$a_j(12)$	0	0	0.5	0	$0.41\overline{6}$	$1.\overline{33}$	2	1	3	$3.1\overline{6}$	4	$4.41\overline{6}$
$n_j(12)$	$4.91\overline{6}$	$3.58\overline{3}$	$2.\overline{33}$	$3.41\overline{6}$	3	$2.41\overline{6}$	2	2	1.75	1	0	0
$E_j(12)$	–	1	1	1	1	2	3	2	4	4	5	–
$L_j(12)$	–	1	3	2	2	3	3	3	4	4	5	–

4.2. Exact solution procedures for SALBP-2

While a large variety of exact solution procedures exist for SALBP-1, only a few have been developed which directly solve SALBP-2. Most research has been devoted to considering search methods which are based on repeatedly solving SALBP-1.

4.2.1. Iterated search methods

Following our argumentation in Section 2, SALBP-2 (minimize c for given m) can be formulated as the problem of finding the smallest cycle time c for which the corresponding SALBP-F instance (m, c) has a feasible solution. Hence, SALBP-2 can be solved by successively solving SALBP-F instances with m stations and various trial cycle times. The latter are restricted to an interval $[\mathrm{LC}, \mathrm{UC}]$ which is bounded by a lower and an upper bound on the cycle time. While lower bounds may be computed by means of the arguments described in Section 4.1, upper bounds are derived from any heuristic solution for SALBP-2 (cf. Section 5).

Any exact procedure for SALBP-1 (cf. Section 3) can easily be modified for SALBP-F instances (m, c) by considering the given cycle time c and by excluding (fathoming) all solutions which have a larger number of stations than m. This is achieved by setting $\mathrm{UM} = m + 1$.

The efficiency of solving a SALBP-2 instance depends on the sequence in which the trial cycle times are examined. Therefore, several search methods have been proposed and tested by, e.g., Dar-El and Rubinovitch (1979), Hackman et al. (1989), Klein and Scholl (1996). Two common methods are the following:

- *Lower bound search*. Starting with the lower bound LC, the trial cycle time c is successively increased by 1 until the respective SALBP-F instance (m, c) is feasible.
- *Binary search*. The search interval $[\mathrm{LC}, \mathrm{UC}]$ is successively subdivided into two subintervals by choosing the mean element $c = \lfloor(\mathrm{LC} + \mathrm{UC})/2\rfloor$. If SALBP-F is feasible for c, the upper bound UC is set to the maximal station time in the corresponding solution. Otherwise, LC is set to $c + 1$. The search stops with the optimal cycle time UC when $\mathrm{UC} = \mathrm{LC}$.

Two further search methods are the Fibonacci binary search and the upper bound search. Computational experiments indicate that the binary search is the best compromise in case of restricted computation time (cf. Scholl, 1999, p. 256).

When the SALBP-F instances are solved heuristically, the whole search procedure becomes a heuristic (cf. Section 5.1).

4.2.2. Direct solution approaches

In recent years, only two B&B procedures which directly solve SALBP-2 have been developed.

- Scholl (1994) proposes the task-oriented B&B procedure TBB-2 which is based on a successive reduction of the precedence graph to a station graph (for extensions see Scholl, 1999, Chapter 4.2.3). At the beginning, the precedence graph is complemented by an initial station graph containing an empty node N_k for the stations $k = 1, \ldots, m$. The nodes N_1 and N_m serve as fictitious source and sink node of the combined graph, respectively. Each task which is assigned to a station k by branching, prefixing or another logical test is merged into the station node N_k. A complete solution is obtained when only the station graph remains. The graph representation of partial solutions allows for applying logical tests which reduce the B&B tree greatly.
- Klein and Scholl (1996) develop an adaptation of SALOME-1 to SALBP-2 which is called SALOME-2. It also utilizes the LLBM, a bidirectional branching strategy, and several dominance and reduction rules which have to be modified to fit the conditions of SALBP-2. A distributed version of SALOME-2 is examined by Bock (2000, Chapter 6.2).

In opposite to SALOME-1, more than two classes of subproblems have to be considered in each node of the enumeration tree depending on the current value of the local lower bound LLC on the cycle time. After examining all maximal station loads feasible for the lower bound cycle time LLC, its value is increased such that at least one

further load becomes feasible. This process of branching a node terminates when a feasible complete solutions with cycle time LLC has been found or no further load is available.

Computational experiments indicate that SALOME-2 gets much better results than TBB-2 when applied to standard benchmark data sets (cf. Scholl, 1999, p. 258). Comparing SALOME-2 with a binary search applying SALOME-1 to each trial cycle time shows that the search method is competitive (cf. Klein and Scholl, 1996). This gives a certain foundation for the concentration on finding effective procedures for SALBP-1 in the past 50 years though SALBP-2 is a problem which may even have the greater practical relevance, because it arises whenever an existing line has to be rebalanced, while SALBP-1 is relevant mainly in case of the first installation of a line.

4.3. Search methods for SALBP-E

As is the case with SALBP-2, instances of the more general SALBP-E may be solved by some search method. Such a method has to find a feasible combination (m, c) of the number m of stations and the cycle time c such that the line efficiency is maximized or, equivalently, the required line capacity $T = m \cdot c$ is minimized (cf. Section 2).

A SALBP-E instance is defined by an interval $[c_{\min}, c_{\max}]$ of possible cycle times and/or an interval $[m_{\min}, m_{\max}]$ of possible numbers of stations. The intervals have to observe the feasibility conditions $T = m \cdot c \geqslant t_{\text{sum}}$ and $c_{\min} \geqslant t_{\max}$ as well as space and productivity constraints.

An obvious solution procedure for SALBP-E consists of considering all possible SALBP-F instances (=(m, c)-combinations with $m \in [m_{\min}, m_{\max}]$ and $c \in [c_{\min}, c_{\max}]$) and examining their feasibility. A feasible combination with minimal value of T is optimal. This may be done with modified procedures for SALBP-1 or SALBP-2, respectively. However, usually a lot of (m, c)-combinations exist. Furthermore, the SALBP-F instances to be considered are NP-complete. Thus, the outlined approach is usually very inefficient.

In order to find an optimal (m, c)-combination more efficiently, search methods similar to those defined for SALBP-2 (Section 4.2) have been examined by Zäpfel (1975, p. 53), Klenke (1977, p. 47) as well as Scholl and Voß (1995). A similar approach is given for a cost-oriented problem by Rosenblatt and Carlson (1985).

The *lower bound search* of Scholl and Voß (1995) starts with a list L of combinations $(m, \text{LC}(m))$ with $m \in [m_{\min}, m_{\max}]$. That is, for each feasible value of m the smallest possible cycle time (given by a lower bound $\text{LC}(m)$; cf. Section 4.1) is considered. In case of $\text{LC}(m) > c_{\max}$, m is removed from the station interval. If $\text{LC}(m) < c_{\min}$, we have to use $\text{LC}(m) = c_{\min}$. The (m, c)-combinations in L are sorted in non-decreasing order of their line capacities $T = m \cdot c$. The search examines the SALBP-F instances in this order, where ties are broken in favor of the smaller m because the corresponding SALBP-F instances are often easier to solve than those with larger m. If the combination (m, c) is identified as being infeasible, m is removed from the station interval in case of $c = c_{\max}$. Otherwise, the next possible cycle time c' (normally $c' := c + 1$) is defined for m and the resulting combination is sorted into the list L. This process continues until a feasible (=optimal) combination is found.

Examination of the SALBP-F instances may be done by applying a (correspondingly modified) SALBP-1 or SALBP-2 procedure. The latter have the advantage that larger increments than 1 may be identified accelerating the search (cf. Scholl, 1999, Chapter 4.3). Furthermore, they more often find feasible solutions in case of restricted computation time (cf. Scholl, 1999, p. 272).

Procedures directly solving SALBP-E are not available. This may be due to the difficulty to direct the search in promising regions of the solution space when neither m nor c is fixed.

5. Heuristic approaches for different versions of SALBP

A large variety of heuristic approaches to different versions of SALBP have been proposed in the last decades. While constructive procedures constructing one or more feasible solution(s) were developed until the mid nineties, improvement procedures using metastrategies like tabu search

and genetic algorithms have been in the focus of researchers in the last decade.

5.1. Constructive procedures

The majority of constructive procedures have been proposed for SALBP-1 and are based on priority rules, others are restricted enumerative procedures. The most recent comprehensive surveys of those approaches are given by Talbot et al. (1986) and Scholl (1999, Chapter 5.1.1). Furthermore, see Boctor (1995) and Ponnambalam et al. (2000).

5.1.1. Priority rule based procedures for SALBP-1

Those procedures use priority values computed for the different tasks based on the task times and the precedence relations given. Some of the most effective ones are given in Table 8 (cf. Talbot et al., 1986; Hackman et al., 1989; Scholl and Voß, 1996). In any case, the tasks are sorted according to non-increasing priority values to get a *priority list*.

By analogy with exact solution procedures (cf. Section 3.2), two *construction schemes* are relevant for priority rule based approaches. They differ with respect to the manner in which the tasks to be assigned are selected out of the set of available tasks (cf. Talbot et al., 1986; Hackman et al., 1989; Scholl and Voß, 1996):

- *Station-oriented procedures*. They start with the first station ($k = 1$). The following stations are considered successively. In each iteration, a task with highest priority which is assignable to the current station k is selected and assigned. When station k is loaded maximally, it is closed, and the next station $k + 1$ is opened.

Table 8
Priority values

Name	Priority value
MaxT	Task time t_j
MaxPW	Positional weight $pw_j = t_j + \sum_{h \in F_j} t_h$
MaxF	Number of followers $\mid F_j^* \mid$
MaxTL	Task time over latest station t_j/L_j
MaxTS	Task time over slack $t_j/(L_j - E_j + 1)$
MaxCPW	Cumulated positional weight $pw_j^* = t_j + \sum_{h \in F_j^*} pw_h^*$

  For the rule MaxPW, this procedure is called *ranked positional weight technique* by Helgeson and Birnie (1961).
- *Task-oriented procedures*. Among all available tasks, one with highest priority is chosen and assigned to the earliest station to which it is assignable.
  Depending on whether the set of available tasks is updated immediately after assigning a task or after assigning all currently available tasks, task-oriented methods can be subdivided into *immediate-update-first* and *general-first-fit* methods (cf. Wee and Magazine, 1982; Hackman et al., 1989).

Theoretical analyses show that both schemes obtain the same solution when the used priority rule is *strongly monotonous*, i.e., the priority value of any task j is smaller than that of each predecessor $h \in P_j$. This is, e.g., true for MaxPW, MaxF, and MaxCPW (cf. Scholl, 1999, p. 182). Computational experiments indicate that, in general, station-oriented procedures get better results than task-oriented ones though no theoretical dominance exists (cf. Scholl and Voß, 1996). This supports the analogous finding in case of exact procedures (cf. Section 3.6).

These classical priority rule based procedures work unidirectionally in forward direction and construct a single feasible solution. Improvements are obtained by following approaches.

- *Flexible bidirectional construction*. The stations to be loaded are considered in forward and backward direction, simultaneously (Scholl and Voß, 1996). That is, a station-oriented procedure considers the earliest and the latest unloaded station at a time. Besides selecting a (forward or backward assignable) task by some priority rule the (earliest or latest) station to be considered next is chosen. Task-oriented procedures simultaneously consider forward and backward available tasks and always choose the one with highest priority. Both approaches require defining reversed priority rules (cf. Scholl, 1999, p. 184).
- *Dynamic priority rules* iteratively adapt the priorities depending on the current partial solutions (cf. Boctor, 1995; Scholl and Voß, 1996). For example, MaxTS can be applied dynamically (in a uni- or

bidirectional procedure) by modifying the earliest and latest stations according to the assignments made.

• *Multi-pass heuristics* repeatedly apply different or stochastic priority rules in order to find several solutions the best of which is taken (cf. Talbot et al., 1986; Arcus, 1966; Bonney et al., 1976; Schofield, 1979).

• *Flexible rule application*. Such procedures try to identify priority rules best suited to solving a certain problem instance. This is done randomly, on the basis of experiences with former rule applications and by exploiting problem structures (cf., e.g., Tonge, 1965; Bennett and Byrd, 1976; Görke and Lentes, 1976; Raouf et al., 1980).

• *Reduction techniques*. Baybars (1986b) proposes a priority based procedure which involves heuristically reducing the problem size by some logical tests. Furthermore, see Tonge (1960), Freeman and Swain (1986), and Fleszar and Hindi (2003).

• *Combined solutions*. As stated in Section 3.5, SALBP-1 can be interpreted as a shortest path problem with exponential numbers of nodes and arcs. Each feasible solution can be represented by a path in such a graph. Therefore, Pinto et al. (1978) describe a two-stage solution approach. In the first step, a number of feasible solutions is determined by a multi-pass heuristic. These solutions are used to construct a subgraph of the complete graph in the second step of the procedure. For this subgraph, a shortest path problem is solved. That is, the outlined approach tries to combine parts of several feasible solutions in order to obtain an improved complete solution.

5.1.2. Incomplete enumeration procedures

A class of heuristic procedures (for SALBP-1 or SALBP-2) consists of incomplete enumeration techniques. Generally, such methods are based on exact enumeration schemes (cf. Section 3), which are modified by heuristically restricting the search space.

• *Heuristic of Hoffmann* (1963). The procedure works unidirectionally in a station-oriented manner. In each iteration $k = 1,2,\ldots$ a load with minimal idle time is generated for station k. That is, a single branch of a station-oriented B&B procedure is constructed.

Nevertheless, it may require considerable computation times, because it has to examine all possible station loads of a current subproblem. Therefore, Gehrlein and Patterson (1975, 1978) propose a modification of the procedure which accepts a load for the currently considered station if a certain amount of idle time is not exceeded. The accepted portion of idle time depends on the balance delay time (total available idle time) for the theoretical minimum number LM1 of stations and can be controlled by a parameter.

An extension of the Hoffmann heuristic which works bidirectionally is proposed by Fleszar and Hindi (2003). This heuristic is combined with a number of bound arguments and reduction techniques (cf. Sections 3.1 and 3.4) and, thus, has become one of the most effective available heuristics for SALBP-1.

• *Truncated enumeration*. Each B&B (or DP) procedure can be applied as a heuristic by adding heuristic fathoming rules or imposing a time limit. For example, subproblems may be fathomed if it is not likely that the current incumbent solution can be improved. This may be examined by comparing the average idle time of already loaded stations with the average idle time being available in an improved solution to be found. Furthermore, the number of descending nodes chosen for branching in a current subproblem can be restricted or partial solutions can be completed heuristically (cf. Hackman et al., 1989, Scholl, 1999, p. 126).

5.1.3. Search methods for SALBP-2 and SALBP-E

As already mentioned in Sections 4.2 and 4.3, solutions for SALBP-2 or SALBP-E may be obtained by iteratively solving SALBP-F instances through SALBP-1 procedures. If heuristic procedures are applied within such search methods, one gets heuristic solutions for SALBP-2 or SALBP-E. Respective procedures are described, among others, by Hackman et al. (1989), Scholl and Voß (1996), and Ugurdag et al. (1997).

5.2. Genetic algorithms

Genetic algorithms (GA) are a general concept for solving complex optimization problems which

is based on manipulating a population of solutions by genetic operators like selection, recombination and mutation (cf. Goldberg, 1989). In order to adapt the general approach to SALBP (or a generalized ALBP), two main difficulties have to be resolved:

• For manipulating solutions by means of genetic operators, they have to be encoded in form of so-called *chromosomes* each of which consists of a sequence of *genes*. Several encoding schemes are possible each having pros and cons concerning the type of applicable genetic operators. In particular, pertaining feasibility of manipulated solutions is a critical issue.

• The objective function of SALBP-1 is not operational for guiding the search to promising parts of the solution space, because it does not give a strong distinction between the solutions' fitness: Usually there are a few optimal solutions which require the minimal number m^* of stations and many others "around them" most of which may require $m^* + 1$ (or some more) stations. That is, a population might consist of solutions all having the same or a few different objective value(s) such that selecting the most promising ones is not obvious. This problem is usually less relevant for SALBP-2 or SALBP-E. Because the GA procedures proposed for different SALBP versions and generalized problems in the literature do not differ in many aspects, we summarize the approaches concerning their resolution of the above mentioned difficulties.

5.2.1. Encoding schemes and genetic operators

• *Standard encoding*. The chromosome is defined as a vector containing the labels of the stations to which the tasks $1,\ldots,n$ are assigned (Anderson and Ferris, 1994, Kim et al., 2000). When standard crossovers or mutations are applied to such chromosomes, the resulting solutions are often infeasible. This aspect must be dealt with by penalizing infeasibilities or rearranging the solution by certain heuristic strategies. Kim et al. (2000) achieve populations without infeasible solutions by decoding chromosomes using a procedure similar to that of Helgeson and Birnie (1961).

Example. For our SALBP-1 example with the precedence graph of Fig. 1 and $c = 10$, two chromosomes representing feasible solutions may be $C_1 = \langle 1,2|3,3,4,4,5,5,6,7\rangle$ and $C_2 = \langle 2,3|1,1,2,4,3,4,5,6\rangle$. Applying a one-point crossover which cuts both chromosomes after the second task and combines them crosswise yields $C_3 = \langle 1,2,1,1,2,4,3,4,5,6\rangle$ and $C_4 = \langle 2,3,3,3,4,4,5,5,6,7\rangle$, two highly infeasible solutions.

• *Order encoding*. The chromosomes are defined as precedence feasible sequences of tasks (cf. Leu et al., 1994; Ajenblit and Wainwright, 1998; Sabuncuoglu et al., 2000; Thilakawardana et al., special issue). They are decoded to feasible SALBP-1 solutions, e.g., by applying the task- or station-oriented construction scheme (cf. Section 3.2) following the encoded sequence of tasks. However, the mapping is not unique, because several feasible sequences may lead to the same solution. Furthermore, several solutions may be derived from one sequence by constructing solutions in different directions (forwards, backwards, bidirectionally; cf. Ajenblit and Wainwright, 1998).

The recombination of chromosomes C_1 and C_2 is usually performed by a *two-point order crossover*, which cuts each of them into three parts (cf. Leu et al., 1994). One offspring C_3 keeps the first and the last part of C_1. The middle part of the sequence is filled by adding the missing tasks in the order in which they are contained in C_2. The other offspring C_4 is built analogously based on the first and the last part of C_2 and added tasks of C_1. Both offspring are feasible due to filling in the middle part in a precedence feasible order. Further crossover and mutation operators are described by Rubinovitz and Levitin (1995) and Thilakawardana et al. (special issue).

Example. Let $C_1 = \langle 1,2,3|4,5,6,7|8,9,10\rangle$ and $C_2 = \langle 3,1,2|4,7,5,6|8,9,10\rangle$ be two feasible sequences for our example instance. Decoding them with the task-oriented construction scheme (with maximal load rule) they represent the solutions $(\{1\},\{2,7\},\{3,4\},\{5,6\},\{8\},\{9\},\{10\})$ with 7 stations and $(\{3,4\},\{1,5\},\{2,7\},\{6,8\},\{9\},\{10\})$ with 6 stations. Cutting them as indicated by vertical lines and recombining them by the two-point crossover results in $C_3 = \langle 1,2,3,4,7,5,6,8,9,10\rangle$ and $C_4 = \langle 3,1,2,4,5,6,7,8,9,10\rangle$. Though the sequences are different from the mates, $C_3(C_4)$ corresponds to the same solution as $C_1(C_2)$. As a

consequence, there is no real change in genetic information.

• *Group encoding*. The encoding of each solution consists of two parts (cf. Falkenauer and Delchambre, 1992, Falkenauer, 1996, 1997; Rekiek et al., 2001, 2002a): The first one is task-oriented and identical to the standard encoding. The second one is group-oriented, i.e., it contains a gene for each station. The genetic operators are only applied to the second part in order to get a more direct influence on the structure of a solution. For reproduction a *modified two-point crossover* is used. The middle part of the second parent C_2 is injected into the first parent C_1 after the first crossing position leading to duplicates of tasks. Each original station of C_1 containing such duplicates and/or tasks which violate precedence constraints are removed. Now some tasks may be missing completely and are reassigned by some heuristic (priority rule based) procedure. The *mutation operator* randomly deletes some stations and reassigns the tasks accordingly.

Example. We consider chromosomes $C_1 = \langle 1,2,3,3,4,4,5,5,6,7{:}1,2,3|4,5|6,7\rangle$ for a solution with 7 stations labeled by numbers 1–7 and $C_2 = \langle b,c,a,a,b,d,c,d,e,f{:}a,b|c,d|e,f\rangle$ for a solution with 6 stations labeled by lower case letters *a*–*f*. The colons separate the two parts of the encodings; the vertical lines indicate the crossing positions, respectively. Injecting $\langle c,d\rangle$ into C_1 results in $\langle 1,2/c,3,3,4,4/d,5/c,5/d,6,7{:}1,2,3,c,d,4,5,\ 6,7\rangle$ where the tasks 2, 6, 7, and 8 are contained twice: $S_1 = \{1\}$, $S_2 = \{2\}$, $S_3 = \{3,4\}$, $S_c = \{2,7\}$, $S_d = \{6,8\}$, $S_4 = \{5,6\}$, $S_5 = \{7,8\}$, $S_6 = \{9\}$, $S_7 = \{10\}$. Deleting the stations 2, 4, and 5 results in $\langle 1,c,3,3,?,d,c,d,6,7{:}1,3,c,d,6,7\rangle$ with task 5 assigned not any longer. Reassigning it by the task-oriented construction scheme leads to $\langle 1,c,3,3,x,d,c,d,6,7{:}1,3,x,c,d,6,7\rangle$ which requires an additional station x and corresponds to the line balance: $S_1 = \{1\}$, $S_3 = \{3,4\}$, $S_x = \{5\}$, $S_c = \{2,7\}$, $S_d = \{6,8\}$, $S_6 = \{9\}$, $S_7 = \{10\}$.

A similar group oriented encoding is used by Suresh et al. (1996) who apply a one-point crossover with subsequent rearrangements of the solutions to achieve feasibility.

• *Indirect encodings*. Further encodings represent the solutions in an indirect manner by coding priority values of tasks (cf. Goncalves and Almeida, 2002) or a sequence of priority rules and corresponding construction schemes to be applied for generating (decoding) the solutions (cf. Bautista et al., 2000; Baykasoglu et al., 2002). This has the advantage that feasibility can be achieved without difficulties. Furthermore, local search heuristics may be applied to improve on the current solution as a means of increasing the fitness, i.e., only local optima are contained in the population (cf. Goncalves and Almeida, 2002).

An encoding which is based on the degree of separation of two adjacent tasks h and j in the precedence graph (0 = assigned to the same station, 1 = assigned to two adjacent stations, 2 = assigned to non-adjacent stations) is proposed by Watanabe et al. (1995). For reproduction a simple 1-point crossover is used. The possible infeasibility of solutions is handled via the fitness function (see below).

5.2.2. Fitness functions

In order to get a more diversified evaluation of a solution's fitness than by using the number of stations or the cycle time required, several approaches have been presented. They are based on measuring the distribution of station times (vertical balancing; cf. Section 2):

- Falkenauer and Delchambre (1992) and Falkenauer (1996) define the fitness $f(S)$ of an SALBP-1 solution $S = (S_1,\ldots,S_{\mathrm{UM}})$ which requires UM stations as a mean score measuring the squared average relative deviation from a full station load:

$$f(S) = \frac{\sum_{k=1}^{\mathrm{UM}} \left(t(S_k)/c\right)^2}{\mathrm{UM}}. \tag{6}$$

- Sabuncuoglu et al. (2000) use the following fitness function, where $tS_{\max}$ denotes the maximal station time ($tS_{\max} \leqslant c$). The first part aims at reducing the imbalance, the second one at minimizing the number of stations (SALBP-1). The factor 2 is set arbitrarily.

$$f(S) = 2\sqrt{\frac{\sum_{k=1}^{UM}(tS_{max} - t(S_k))^2}{UM}} + \frac{\sum_{k=1}^{UM}(tS_{max} - t(S_k))}{UM}. \quad (7)$$

- Bautista et al. (2000) propose the following fitness function for SALBP-1 which considers the degree of imbalance in the first term and the absolute deviation of the number UM of stations used to the lower bound LM1:

$$f(S) = \frac{\sqrt{\sum_{k=1}^{UM}(c - t(S_k))^2}}{c \cdot \sqrt{UM}} + LM1 - UM. \quad (8)$$

- Anderson and Ferris (1994) construct a GA for SALBP-2 and use a fitness function which sums up the maximal station time (defining the realized cycle time) and a penalty term for precedence violations. In order to minimize the cycle time, the fitness function is to be minimized, too. The GA of Watanabe et al. (1995) is also designed for SALBP-2 and takes the line efficiency reduced by a penalty term for infeasible surplus stations as fitness values.

5.2.3. Further components and computational results

Besides the encoding schemes and the fitness functions, all GA procedures proposed for SALBP-1 or SALBP-2 in the literature (see the references given above) apply standard approaches for generating the initial population, setting crossover and mutation probabilities, etc. Unfortunately, the computational testing of most GA has been done ignoring existing test beds and state-of-the-art solution methods or using the most simple test data available such that most results are not meaningful. At least, GA seem to be competitive to the best known constructive methods (cf. Sabuncuoglu et al., 2000; Thilakawardana et al., special issue) and some type of tabu search (cf. Goncalves and Almeida, 2002; Baykasoglu et al., 2002). Furthermore, they allow for including additional problem characteristics like assignment constraints or further objectives (cf. Bautista et al., 2000; Ponnambalam et al., 2000) and may be combined with other heuristics in hybrid approaches (cf. Goncalves and Almeida, 2002; Falkenauer, 1996).

5.3. Local search and metastrategies

Local search (or improvement) procedures try to improve a given feasible solution by iteratively *transforming* it into other feasible solutions. Such transformations are referred to as *moves*. Solutions which may be obtained from a given solution S by means of a single move are called neighbouring solutions or *neighbourhood* of S. Traditionally, local search heuristics try to find a sequence of moves which produces a trajectory of successively improved solutions and terminate in a *local optimum* which might be far from optimality.

This difficulty is overcome by modern metastrategies like *tabu search* (TS; cf. Glover and Laguna, 1997) and *simulated annealing* (SA; cf. Aarts and Korst, 1989). In the following, we discuss the main components of such procedures when applied to SALBP-1 and SALBP-2.

5.3.1. Moves and neighborhood definition

All local search procedures for SALBP are based on shifts and swaps (cf. Moodie and Young, 1965; Rachamadugu and Talbot, 1991), which can be explained using the following notation:

LP_j: latest station to which a predecessor of task j is currently assigned.
ES_j: earliest station to which a successor of task j is currently assigned.

Two types of moves are relevant in local search procedures for SALBP:

- A *shift* (j, k_1, k_2) describes the movement of a task j from station k_1 to station k_2 with $k_1 \neq k_2$. This move is feasible if $k_2 \in [LP_j, ES_j]$.
- A *swap* (j_1, k_1, j_2, k_2) exchanges tasks j_1 and j_2, which are not related by precedence, between different stations k_1 and k_2. This move is feasible if the two corresponding shifts (j_1, k_1, k_2) and (j_2, k_2, k_1) are feasible.

In order to restrict our presentation to swap moves, we interpret shifts as swaps which exchange a real task j_1 and a fictitious task j_2 with zero operation time and no precedence relations.

Two basic local search strategies are *best fit* (find and perform the most improving move) and *first fit* (perform the first improving move found).

5.3.2. Tabu search for SALBP-2

An elementary TS procedure for SALBP-2 is proposed by Heinrici (1994). Scholl and Voß (1996) develop and thoroughly examine a TS procedure with a lot of optional components. We restrict the presentation to the essential features:

Initial solution. It may be determined by any constructive procedure for SALBP-2 or randomly (cf. Section 5.1). Experiments show that the quality of the initial solution is not very important for the overall solution quality (cf. Scholl, 1999, p. 263).

Local search strategy. The best fit strategy is applied, i.e., in each iteration, the most improving or least deteriorating (if no improving is available) move is performed.

Tabu management. In order to avoid cycling, attributes of moves just performed are set tabu for a number of iterations TD (=tabu duration) and stored in a tabu list TL (recency based memory). When a swap (j_1,k_1,j_2,k_2) is performed the attributes (j_1,k_1) and (j_2,k_2) are added to TL such that removing j_1 to k_1 and j_2 to k_2 is temporarily forbidden for TD iterations. Since TD is the most critical parameter concerning the performance of (static) TS, different strategies have been tested by Scholl and Voß (1996) showing that a dynamic modification of TD (randomly or systematically) is better than using a fixed value. Furthermore, a moving gap strategy which cyclically activates only parts of the tabu list at a time is recommendable. In case of a move which leads to an improved overall solution, the tabu status is ignored (global aspiration criterion).

Stopping criteria. The search is stopped whenever a prespecified number of iterations have been performed, a prespecified time limit is reached or the solution is identified as being optimal by means of lower bounds.

Intensification and diversification. In order to intensify the search in certain regions or to direct the search into yet unvisited parts of the solution space, a frequency based memory is used. In this memory, the relative number of iterations, any task j belongs to any station k, is stored (denoted as z_{jk}). Several phases of the search are either used for collecting frequency information, fixing tasks j in a station k where they have a high z_{jk} value (intensification) or avoiding that tasks j reenter a station k where they have a high z_{jk} value (diversification).

Conflict management. A conflict occurs when no admissible move is available, i.e., the attributes stored in TL prevent all possible moves from being performed. Among several strategies, Scholl and Voß (1996) recommend one which systematically moves predecessors and successors away from *critical tasks* (those assigned to stations with maximal station time).

Computational experiments of Scholl and Voß (1996) show that the TS procedure outlined above gets much better results than constructive procedures. In particular, it is well suited to finding improved initial upper bounds for B&B procedures (see Section 4.2) such that their performance is improved. A similar result has been obtained by Scholl et al. (1997) for the related bin packing problem.

An extension of the described TS procedure to a mixed-model balancing problem with additional objectives is given by Pastor et al. (2002).

5.3.3. Tabu search for SALBP-1

Developing a TS procedure for SALBP-1 is not as straightforward as in case of SALBP-2. This is due to the fact that only three situations can occur after a move (cf. our discussion at the beginning of Section 5.2): (1) the number m of stations is unchanged (swapping two real tasks), (2) an additional station m is required (shirting a task in an empty station), (3) one station is empty (shifting the only task in this station to another one). No problem arises in case (3). However, in most iterations a large number of (1) or (2) moves have to be evaluated which have only two different objective function values m and $m+1$. In this situation finding a promising search direction is rather complicated.

Scholl and Voß (1996) conclude that applying their TS procedure for SALBP-2 within a lower bound search (called dual strategy; cf. Section 4.2) is the best way out of this dilemma.

Chiang (1998) proposes a TS procedure similar to the SALBP-2 approach of Scholl and Voß (1996) but uses a surrogate objective function that maximizes the sum over the squared station times. While minimizing the number of stations, it additionally favors solutions containing some heavily loaded stations to those solutions having more smoothly loaded ones. This effect successively directs the search to solutions where saving a station in a single move is probable.

Computational experiments indicate that both approaches are successful on principle. However, Chiang (1998) reports only limited results for the simplest data set on hand which are not very meaningful. Scholl and Voß (1996) find out that their dual strategy is competitive to exact procedures (applied as a restricted enumeration) in case of short computation times but is not superior to SALOME-1 in finding good feasible solutions quickly. In opposite to SALBP-2, the quality of the initial solution seems to be important for the quality of the best solution found.

A further TS procedure for SALBP-1 is proposed by Lapierre et al. (this issue) and tested on an arbitrary subset of the test problems available. For these instances it compares favorably with Chiang's approach.

5.3.4. Simulated annealing (SA) procedures

Heinrici (1994) proposes an SA procedure for SALBP-2 which is based on shifts and swaps. An SA approach for a stochastic variant of SALBP-1 is proposed by Suresh and Sahu (1994). McMullen and Frazier (1998) propose a SA procedure for a generalization of SALBP-1 with respect to parallel stations, stochastic task times and alternative objectives.

5.3.5. LP-based improvement procedure

Ugurdag et al. (1997) develop a two-stage heuristic for SALBP-2 (with additional vertical balancing) which is based on an integer program. The procedure starts with a priority rule based search method that generates one or more initial solutions. The second stage improves on the initial solution by a simplex-like improvement procedure solving a relaxation of the mathematical model with the nice property of having only integer extreme points.

5.3.6. Ant colony optimization approach

Bautista and Pereira (2002) present an ant algorithm for SALBP-1 which is based on priority rule based procedures. McMullen and Tarasewich (2003) propose an ant algorithm for a generalization of SALBP with respect to parallel stations, stochastic task times, multiple objectives and mixed-model production.

6. Conclusions and further research

The survey given shows that research has made significant algorithmic developments in solving simple assembly line balancing problems (SALBP). Though SALBP is a class of NP-hard optimization problems, effective exact and heuristic procedures are available which solve medium-sized instances in a quality sufficient for use in practice. However, further algorithmic improvement is necessary for solving large-scale instances.

More recently, assembly line balancing research evolved towards formulating and solving generalized problems (GALBP) with different additional characteristics such as cost functions, equipment selection, paralleling, U-shaped line layout and mixed-model production. Reviewing the literature on GALBP (cf. Becker and Scholl, this issue) shows that a lot of relevant problems have been identified and modelled but development of sophisticated solution procedures has just begun. Thus, additional research is necessary to adopt state-of-the-art solution concepts for SALBP to the variety of GALBP. Furthermore, standardized and realistic test beds are required for testing and comparing methodical enhancements.

Despite of having available effective solution procedures for different assembly line balancing problems their use in practice is limited. Besides imposing restrictive assumptions for defining computationally tractable models, this is due to a lack

of commercial software that includes state-of-the-art solution procedures.

References

Aarts, E.H.L., Korst, J.H.M., 1989. Simulated annealing and Boltzmann machines: A stochastic approach to combinatorial optimization and neural computing. Wiley, Chichester.

Ajenblit, D.A., Wainwright, R.L., 1998. Applying genetic algorithms to the U-shaped assembly line balancing problem. In: Proceedings of the 1998 IEEE International Conference on Evolutionary Computation, Anchorage, AK, pp. 96–101.

Alvim, A.C.F., Ribeiro, C.C., Glover, F., Aloise, D.J., 2003. A hybrid improvement heuristic for the one-dimensional bin packing problem. Working Paper, Catholic University of Rio de Janeiro, Brazil.

Anderson, E.J., Ferris, M.C., 1994. Genetic algorithms for combinatorial optimization: The assembly line balancing problem. ORSA Journal on Computing 6, 161–173.

Arcus, A.L., 1966. COMSOAL: A computer method of sequencing operations for assembly lines. International Journal of Production Research 4, 259–277.

Balas, E., 1965. An additive algorithm for solving linear programs with zero–one variables. Operations Research 13, 517–546.

Bautista, J., Pereira, J., 2002. Ant algorithms for assembly line balancing. In: Dorigo, M., Di Caro, G., Sampels, M. (Eds.), Ant Algorithms, Third International Workshop, ANTS 2002, Brussels, Belgium, 2002, Proceedings, Lecture Notes in Computer Science, 2463. Springer, Berlin, pp. 65–75.

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

Baybars, I., 1986a. A survey of exact algorithms for the simple assembly line balancing problem. Management Science 32, 909–932.

Baybars, I., 1986b. An efficient heuristic method for the simple assembly line balancing problem. International Journal of Production Research 24, 149–166.

Baykasoglu, A., Özbakir, L., Telcioglu, M.B., 2002. Multiple-rule based genetic algorithm for simple assembly line balancing problems (MRGA-SALBP-I). In: Proceedings of the 5th International Conference on Managing Innovations in Manufacturing (MIM) 2002, Milwaukee, WI, USA, pp. 402–408.

Becker, C., Scholl, A., this issue. A survey on problems and methods in generalized assembly line balancing. European Journal of Operations Research. doi:10.1016/j.ejor.2004.07.023.

Bennett, G.B., Byrd, J., 1976. A trainable heuristic procedure for the assembly line balancing problem. AIIE Transactions 8, 195–201.

Berger, I., Bourjolly, J.-M., Laporte, G., 1992. Branch-and-bound algorithms for the multi-product assembly line balancing problem. European Journal of Operational Research 58, 215–222.

Betts, J., Mahmoud, K.I., 1989. A method for assembly line balancing. Engineering Costs and Production Economics 18, 55–64.

Bock, S., 2000. Modelle und verteilte Algorithmen zur Planung getakteter Fließlinien: Ansätze zur Unterstützung eines effizienten Mass Customization, Gabler, Wiesbaden.

Bock, S., Rosenberg, O., 1998. A new distributed fault-tolerant algorithm for the simple assembly line balancing problem. In: Kischka, P. et al. (Eds.), Operations Research Proceedings 1997. Springer, Berlin, pp. 474–480.

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

Boctor, F.F., 1995. A multiple-rule heuristic for assembly line balancing. Journal of the Operational Research Society 46, 62–69.

Bonney, M.C., Schofield, N.A., Green, A., 1976. Assembly line balancing and mixed model line sequencing. In: Proceedings of the Fifth INTERNET World Congress, Birmingham, pp. 112–121.

Buxey, G.M., Slack, N.D., Wild, R., 1973. Production flow line system design—A review. AIIE Transactions 5, 37–48.

Chiang, W.-C., 1998. The application of a tabu search metaheuristic to the assembly line balancing problem. Annals of Operations Research 77, 209–227.

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

De Reyck, B., Herroelen, W., 1997. Assembly line balancing by resource-constrained project scheduling—A critical appraisal. Foundations of Computing and Control Engineering 22, 143–167.

Driscoll, J., Thilakawardana, D., 2001. The definition of assembly line balancing difficulty and evaluation of balance solution quality. Robotics and Computer Integrated Manufacturing 17, S81–S86.

Easton, F., Faaland, B., Klastorin, T.D., Schmitt, T., 1989. Improved network based algorithms for the assembly line balancing problem. International Journal of Production Research 27, 1901–1915.

Erel, E., Sarin, S.C., 1998. A survey of the assembly line balancing procedures. Production Planning and Control 9, 414–434.

Falkenauer, E., 1996. A hybrid grouping algorithm for bin packing. Journal of Heuristics 2, 5–30.

Falkenauer, E., 1997. A grouping genetic algorithm for line balancing with resource dependent task times. In: Proceedings of the Fourth International Conference on Neural Information Processing 1997, University of Otago, Dunedin, New Zealand, pp. 464–468.

Falkenauer, E., Delchambre, A., 1992. A genetic algorithm for bin packing and line balancing. In: Proceedings of the 1992

IEEE International Conference on Robotics and Automation, Nice, France, pp. 1186–1192.
Fekete, S.P., Schepers, J., 2001. New classes of fast lower bounds for bin packing problems. Mathematical Programming 91, 11–31.
Fleszar, K., Hindi, K.S., 2003. An enumerative heuristic and reduction methods for the assembly line balancing problem. European Journal of Operational Research 145, 606–620.
Freeman, D.S., Swain, R.W., 1986. A heuristic technique for balancing automated assembly lines. Research Report No. 86-6, Industrial and Systems Engineering Department, University of Florida, Gainesville.
Gehrlein, W.V., Patterson, J.H., 1975. Sequencing for assembly lines with integer task times. Management Science 21, 1064–1070.
Gehrlein, W.V., Patterson, J.H., 1978. Balancing single model assembly lines: Comments on a paper by E.M. Dar-El (Mansoor). AIIE Transactions 10, 109–112.
Ghosh, S., Gagnon, R.J., 1989. A comprehensive literature review and analysis of the design, balancing and scheduling of assembly systems. International Journal of Production Research 27, 637–670.
Glover, F., Laguna, M., 1997. Tabu search. Kluwer Academic Publishers, Boston.
Görke, M., Lentes, H.-P., 1976. Leistungsabstimmung von Montagelinien. Arbeitsvorbereitung 13, 71–77, 106–113, and 147–153.
Goldberg, D.E., 1989. Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Reading, MA.
Goncalves, J.F., Almeida, J.R., 2002. A hybrid genetic algorithm for assembly line balancing. Journal of Heuristics 8, 629–642.
Gutjahr, A.L., Nemhauser, G.L., 1964. An algorithm for the line balancing problem. Management Science 11, 308–315.
Hackman, S.T., Magazine, M.J., Wee, T.S., 1989. Fast, effective algorithms for simple assembly line balancing problems. Operations Research 37, 916–924.
Heinrici, A., 1994. A comparison between simulated annealing and tabu search with an example from the production planning. In: Dyckhoff, H. et al. (Eds.), Operations Research Proceedings 1994. Springer, Berlin, pp. 498–503.
Held, M., Karp, R.M., Shareshian, R., 1963. Assembly line balancing—Dynamic programming with precedence constraints. Operations Research 11, 442–459.
Helgeson, W.B., Birnie, D.P., 1961. Assembly line balancing using the ranked positional weight technique. Journal of Industrial Engineering 12, 394–398.
Hoffmann, T.R., 1963. Assembly line balancing with a precedence matrix. Management Science 9, 551–562.
Hoffmann, T.R., 1990. Assembly line balancing: A set of challenging problems. International Journal of Production Research 28, 1807–1815.
Hoffmann, T.R., 1992. EUREKA: A hybrid system for assembly line balancing. Management Science 38, 39–47.
Hoffmann, T.R., 1993. Response to note on microcomputer performance of "FABLE" on Hoffmann's data sets. Management Science 39, 1192–1193.
Jackson, J.R., 1956. A computing procedure for a line balancing problem. Management Science 2, 261–271.
Jaeschke, G., 1964. Branching and Bounding: Eine allgemeine Methode zur Lösung kombinatorischer Probleme. Ablauf- und Planungsforschung 5, 133–155.
Johnson, R.V., 1981. Assembly line balancing algorithms: Computation comparisons. International Journal of Production Research 19, 277–287.
Johnson, R.V., 1988. Optimally balancing large assembly lines with "FABLE". Management Science 34, 240–253.
Johnson, R.V., 1993. Note: Microcomputer performance of "FABLE" on Hoffmann's data sets. Management Science 39, 1190–1193.
Kao, E.P.C., Queyranne, M., 1982. On dynamic programming methods for assembly line balancing. Operations Research 30, 375–390.
Kim, Y.K., Kim, Y., Kim, Y.J., 2000. Two-sided assembly line balancing: A genetic algorithm approach. Production Planning and Control 11, 44–53.
Klein, R., Scholl, A., 1996. Maximizing the production rate in simple assembly line balancing—A branch and bound procedure. European Journal of Operational Research 91, 367–385.
Klein, R., Scholl, A., 1999. Computing lower bounds by destructive improvement—An application to resource-constrained project scheduling. European Journal of Operational Research 112, 322–346.
Klein, R., Scholl, A., 2000. PROGRESS: Optimally solving the generalized resource-constrained project scheduling problem. Mathematical Methods of Operations Research 52, 467–488.
Klenke, H., 1977. Ablaufplanung bei Fließfertigung. Gabler, Wiesbaden.
Labbé, M., Laporte, G., Mercure, H., 1991. Capacitated vehicle routing on trees. Operations Research 39, 616–622.
Lapierre, S.D., Ruiz, A., Soriano, P., this issue. Balancing assembly lines with tabu search. European Journal of Operational Research. doi:10.1016/j.ejor.2004.07.031.
Lawler, E.L., 1979. Efficient implementation of dynamic programming algorithms for sequencing problems, Report BW 106/79, Stichting Mathematisch Centrum, Amsterdam.
Leu, Y.Y., Matheson, L.A., Rees, L.P., 1994. Assembly line balancing using genetic algorithms with heuristic-generated initial populations and multiple evaluation criteria. Decision Sciences 25, 581–606.
Martello, S., Toth, P., 1990. Knapsack problems—Algorithms and computer implementations. Wiley, New York.
McMullen, P.R., Frazier, G.V., 1998. Using simulated annealing to solve a multiobjective assembly line balancing problem with parallel workstations. International Journal of Production Research 36, 2717–2741.
McMullen, P.R., Tarasewich, P., 2003. Using ant techniques to solve the assembly line balancing problem. IIE Transactions 35, 605–617.
McNaughton, R., 1959. Scheduling with deadlines and loss functions. Management Science 6, 1–12.

Merengo, C., Nava, F., Pozetti, A., 1999. Balancing and sequencing manual mixed-model assembly lines. International Journal of Production Research 37, 2835–2860.

Mertens, P., 1967. Fließbandabstimmung mit dem Verfahren der begrenzten Enumeration nach Müller-Merbach. Ablauf- und Planungsforschung 8, 429–433.

Moodie, C.L., Young, H.H., 1965. A heuristic method of assembly line balancing for assumptions of constant or variable work element times. Journal of Industrial Engineering 16, 23–29.

Nourie, F.J., Venta, E.R., 1991. Finding optimal line balances with OptPack. Operations Research Letters 10, 165–171.

Nourie, F.J., Venta, E.R., 1996. Microcomputer performance of OptPack on Hoffmann's data sets: Comparison with Eureka and FABLE. Management Science 42, 304–306.

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

Peeters, M., Degraeve, Z., this issue. An linear programming based lower bound for the simple assembly line balancing problem. European Journal of Operational Research. doi:10.1016/j.ejor.2004.07.024.

Pinnoi, A., Wilhelm, W.E., 1997a. A branch and cut approach for workload smoothing on assembly lines. INFORMS Journal on Computing 9, 335–350.

Pinnoi, A., Wilhelm, W.E., 1997b. A family of hierarchical models for assembly system design. International Journal of Production Research 35, 253–280.

Pinnoi, A., Wilhelm, W.E., 1998. Assembly system design: A branch and cut approach. Management Science 44, 103–118.

Pinto, P.A., Dannenbring, D.G., Khumawala, B.M., 1978. A heuristic network procedure for the assembly line balancing problem. Naval Research Logistics Quarterly 25, 299–307.

Ponnambalam, S., Aravindan, G., Mogileeswar Naidu, G., 2000. A multi-objective genetic algorithm for solving assembly line balancing problem. International Journal of Advanced Manufacturing Technology 16, 341–352.

Rachamadugu, R., Talbot, B., 1991. Improving the equality of workload assignments in assembly lines. International Journal of Production Research 29, 619–633.

Raouf, A., Tsui, C.L., El-Sayed, E.A., 1980. A new heuristic approach to assembly line balancing. Computers and Industrial Engineering 4, 223–234.

Rekiek, B., de Lit, P., Pellichero, F., Falkenauer, E., Delchambre, A., 1999. Applying the equal piles problem to balance assembly lines. In: Proceedings of the ISATP 1999, Porto, Portugal, pp. 399–404.

Rekiek, B., de Lit, P., Pellichero, F., L'Eglise, T., Fouda, P., Falkenauer, E., Delchambre, A., 2001. A multiple objective grouping genetic algorithm for assembly line balancing. Journal of Intelligent Manufacturing 12, 467–485.

Rekiek, B., de Lit, P., Delchambre, A., 2002a. Hybrid assembly line design and user's preferences. International Journal of Production Research 40, 1095–1111.

Rekiek, B., Dolgui, A., Delchambre, A., Bratcu, A., 2002b. State of art of optimization methods for assembly line design. Annual Reviews in Control 26, 163–174.

Rosenblatt, M.J., Carlson, R.C., 1985. Designing a production line to maximize profit. IIE Transactions 17, 117–121.

Rubinovitz, J., Levitin, G., 1995. Genetic algorithm for assembly line balancing. International Journal of Production Economics 41, 343–354.

Sabuncuoglu, I., Erel, E., Tanyer, M., 2000. Assembly line balancing using genetic algorithms. Journal of Intelligent Manufacturing 11, 295–310.

Saltzman, M.J., Baybars, I., 1987. A two-process implicit enumeration algorithm for the simple assembly line balancing problem. European Journal of Operational Research 32, 118–129.

Schofield, N.A., 1979. Assembly line balancing and the application of computer techniques. Computers and Industrial Engineering 3, 53–69.

Scholl, A., 1993. Data of assembly line balancing problems. Schriften zur Quantitativen Betriebswirtschaftslehre 16/93, TU Darmstadt.

Scholl, A., 1994. Ein B&B-Verfahren zur Abstimmung von Einprodukt-Fließbändern bei gegebener Stationsanzahl. In: Dyckhoff, H. et al. (Eds.), Operations Research Proceedings 1993, Springer, Berlin, pp. 175–181.

Scholl, A., 1999. Balancing and sequencing assembly lines, 2nd ed. Physica, Heidelberg.

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

Scholl, A., Klein, R., 1999. Balancing assembly lines effectively—A computational comparison. European Journal of Operational Research 114, 50–58.

Scholl, A., Voß, S., 1995. Kapazitätsorientierte Leistungsabstimmung in der Fließfertigung. Zeitschrift für betrie-bswirtschaftliche Forschung 47, 919–935.

Scholl, A., Voß, S., 1996. Simple assembly line balancing—Heuristic approaches. Journal of Heuristics 2, 217–244.

Scholl, A., Klein, R., Jürgens, C., 1997. BISON: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. Computers and Operations Research 24, 627–645.

Schrage, L., Baker, K.R., 1978. Dynamic programming solution of sequencing problems with precedence constraints. Operations Research 26, 444–449.

Shtub, A., Dar-El, E.M., 1989. A methodology for the selection of assembly systems. International Journal of Production Research 27, 175–186.

Sprecher, A., 1999. A competitive branch-and-bound algorithm for the simple assembly line balancing problem. International Journal of Production Research 37, 1787–1816.

Sprecher, A., 2003. Dynamic search tree decomposition for balancing assembly lines by parallel search. International Journal of Production Research 41, 1413–1430.

Suresh, G., Sahu, S., 1994. Stochastic assembly line balancing using simulated annealing. International Journal of Production Research 32, 1801–1810.

Suresh, G., Vinod, V.V., Sahu, S., 1996. Genetic algorithm for assembly line balancing. Production Planning and Control 7, 38–46.

Talbot, F.B., Patterson, J.H., 1984. An integer programming algorithm with network cuts for solving the assembly line balancing problem. Management Science 30, 85–99.

Talbot, F.B., Patterson, J.H., Gehrlein, W.V., 1986. A comparative evaluation of heuristic line balancing techniques. Management Science 32, 430–454.

Thilakawardana, D., Driscoll, J., Deacon, G. An efficient genetic algorithm application in assembly line balancing. European Journal of Operational Research. Balancing of Automated Assembly and Transfer Lines (Special issue).

Tonge, F.M., 1960. Summary of a heuristic line balancing procedure. Management Science 7, 21–39.

Tonge, F.M., 1965. Assembly line balancing using probabilistic combinations of heuristics. Management Science 11, 727–735.

Ugurdag, H.F., Rachamadugu, R., Papachristou, C.A., 1997. Designing paced assembly lines with fixed number of stations. European Journal of Operational Research 102, 488–501.

Van Assche, F., Herroelen, W.S., 1979. An optimal procedure for the single-model deterministic assembly line balancing problem. European Journal of Operational Research 3, 142–149.

Watanabe, T., Hashimoto, Y., Nishikawa, L., Tokumaru, H., 1995. Line balancing using a genetic evolution model. Control Engineering Practice 3, 69–76.

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

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