Journal of Heuristics, 2, 217–244 (1996)
© 1996 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Simple Assembly Line Balancing—Heuristic Approaches

ARMIN SCHOLL
Technische Hochschule Darmstadt, Institut für Betriebswirtschaftslehre, Hochschulstr. 1, D-64289 Darmstadt, Germany

STEFAN VOß
Technische Universität Braunschweig, Abteilung Allg. BWL, Wirtschaftsinformatik und Informationsmanagement, Abt-Jerusalem-Str. 7, D-38106 Braunschweig, Germany

Abstract. In this paper heuristics for Type 1 and Type 2 of the Simple Assembly Line Balancing Problem (SALBP) are described. Type 1 of SALBP (SALBP-1) consists of assigning tasks to work stations such that the number of stations is minimized for a given production rate whereas Type 2 (SALBP-2) is to maximize the production rate, or equivalently, to minimize the sum of idle times for a given number of stations. In both problem types, precedence constraints between the tasks have to be considered.

We describe bidirectional and dynamic extensions to heuristic priority rules widely used for SALBP-1. For the solution of SALBP-2 we present search methods which involve the repetitive application of procedures for SALBP-1. Furthermore, improvement procedures for SALBP-2 are developed and combined with tabu search, a recent strategy to overcome local optimality. Several optional elements of tabu search are discussed. Finally, the application of a nontraditional tabu search approach to solve SALBP-1 is investigated. Computational experiments validate the effectiveness of our new approaches.

Keywords: production/scheduling-line balancing; heuristic; tabu search

1. Introduction

An assembly line consists of a sequence of **m work stations** which are connected by a conveyor belt. Each station repeatedly has to perform a set of tasks on consecutive product units moving along the line at constant speed. Because of the uniform movement of the line each product unit spends the same fixed time interval, called the **cycle time** ***c***, in every work station. As a consequence, the cycle time c determines the **production rate** which is $1/c$. Tasks or operations are indivisible elements of work which have to be performed to assemble a product. The execution of each of ***n*** **tasks** $j = 1, \dots, n$ requires a fixed time interval, the task time t_j which is assumed to be integral. Due to technological restrictions **precedence constraints** partially specifying the sequence of tasks have to be considered. These constraints can be represented by a precedence graph containing nodes for all tasks and arcs (i, j) if task i has to be completed before task j can be started.

The Simple Assembly Line Balancing Problem (SALBP) can be stated as follows (see, e.g., Baybars (1986) and Domschke et al. (1993, chapter 4)). Each task has to be assigned to exactly one station of the assembly line such that no precedence constraint is violated. Equivalently, a partitioning of the set of all tasks into disjoint sets S_k with $k = 1, \dots, m$ has to be determined. For each arc (i, j) of the precedence graph the relation $h \leq k$ must hold if $i \in S_h$ and $j \in S_k$. The station time $t(S_k)$ is equal to the sum of the task times of all tasks which are included in S_k. The maximum station time determines the cycle time c, and all stations k with less station time have an idle time $c - t(S_k)$. The objective is to find a cycle time c and a number m of stations as well as a corresponding task assignment which minimizes the sum of idle times over all stations (i.e. minimize $m \cdot c$ minus the sum of all task times). Since the objective function does not linearly depend on the variables m and c, the following variants of SALBP are often considered:

- SALBP-1: Given the cycle time c, minimize the number m of stations.
- SALBP-2: Given the number m of stations, minimize the cycle time c.

Since the production rate has to be specified as a fixed parameter, SALBP-1 is present when a new assembly line system has to be installed and the external demand can be well estimated. Opposite to this, SALBP-2 leads to maximization of the production rate of an existing assembly line which is important when changes in the production process or the demand structure take place.

SALBP-1 as well as SALBP-2 are NP-hard problems. SALBP-1 reduces to the well-known bin-packing problem and SALBP-2 to the problem of scheduling m identical parallel machines with the objective of minimizing the makespan if precedence constraints are omitted (see, e.g., Baybars (1986)).

In the last four decades a large variety of heuristic and exact solution procedures, especially for SALBP-1, have been proposed in the literature (for a survey see, e.g., Domschke et al. (1993)). The first group of algorithms consists of priority based methods and restricted enumerative procedures (see Talbot et al. (1986)). The second group mainly contains dynamic programming and branch and bound approaches (see Baybars (1986)). The most efficient exact algorithms like FABLE of Johnson (1988) and EUREKA of Hoffmann (1992) use the branch and bound paradigm. If EUREKA is unsuccessful in finding an optimal solution within a specified length of computation time it switches to a heuristic.

Contrary to SALBP-1 only a small number of procedures have been proposed to solve SALBP-2. Most of them use a search method which iteratively solves SALBP-1 instances to obtain solutions of SALBP-2 (see, e.g., Hackman et al. (1989)). With respect to heuristics, however, no comprehensive results derived from any empirical investigation are reported which compare these search methods and improvement procedures. Optimum-seeking methods which directly solve SALBP-2 are proposed by Charlton and Death (1969), Scholl (1994) as well as Klein and Scholl (1996).

The remainder of the paper is organized as follows. In Section 2 we describe unidirectional and bidirectional priority based procedures for SALBP-1 and their iterative application within search procedures for solving SALBP-2. In Section 3 we discuss the basic ideas of tabu search, a metastrategy, which enables improvement procedures to overcome local optimality. First, we apply tabu search as a static approach to SALBP-2 and then discuss some extensions.

A general problem with respect to tabu search is a proper move definition. Applying the natural neighbourhood based on shift and swap moves for SALBP-1 results in a move definition that does not allow for a clear distinction between different solutions. Even when allowing for strategic oscillation there is merely any effect on the objective function value.

With respect to this specific problem we employ an idea of general interest to overcome these difficulties that may be transferred to other problem classes as well.

That is, the tabu search approach for SALBP-2 is combined with a lower bound strategy to devise a nontraditional solution scheme for SALBP-1 which allows for overcoming the boundaries of feasibility in a different way than previously proposed within the context of tabu search. Section 4 describes our results obtained from a large number of computational experiments which indicate the efficiency of bidirectional priority based procedures and tabu search algorithms. Special mention deserves the fact that our tabu search approach for SALBP-1 produces comparative or better solutions than the best known exact procedures FABLE and EUREKA for given time limits. In Section 5 the results of the paper are summarized and some conclusions are given.

2. Priority Based Procedures

Most heuristics for finding initial solutions for SALBP-1 are based on static priority rules which are applied unidirectionally. In Section 2.1 we discuss an extension to bidirectional planning and introduce dynamic priority rules. Section 2.2 describes search methods which solve SALBP-2 by iteratively applying procedures for SALBP-1.

2.1. Unidirectional and Bidirectional Priority Based Methods for SALBP-1

A class of simple heuristic strategies for SALBP-1 uses priority rules to build a task ranking. Following this ordering the tasks are assigned to stations in a forward direction. Such procedures can be easily stated using the following **notation** and the definition of available and assignable tasks.

c	cycle time (in general assumed to be integral)
t_j	task time of task $j = 1, \dots, n$
t_{sum}	sum of all task times $(\Sigma_j t_j)$
$t_{\max}$	maximum task time
P_j	set of tasks which must precede task j immediately (predecessors)
F_j	set of tasks which must follow task j immediately (successors, followers)
P_j^*	set of all tasks which must precede task j
F_j^*	set of all tasks which must follow task j
$\bar{m}$	upper bound on the number of stations (trivial upper bound: $\bar{m} := n$)
$\lceil x \rceil$	smallest integer $\geq x$, $\lfloor x \rfloor$ largest integer $\leq x$
$E_j :=$	$\lceil (t_j + \sum_{h \in P_j^*} t_h)/c \rceil$ earliest station for task j
$L_j :=$	$\bar{m} - \lceil (t_j + \sum_{h \in F_j^*} t_h)/c \rceil$ latest station to which task j can be assigned if a feasible $\bar{m} - 1$ station solution exists
S_k	set of tasks which are currently assigned to station $k = 1, \dots, \bar{m}$
$t(S_k)$	station time of station k, $t(S_k) \leq c$
$s_j :=$	$L_j - E_j$ slack of task j[1]

Definition. An unassigned task j is called *available* if all tasks which must precede j are already assigned. An available task j is *assignable to station k* if all predecessors are assigned to the stations $1, \dots, k$ and if task j fits into station k without exceeding the cycle time. More formally, task j is assignable to station $k \in [E_j, L_j]$ iff $P_j^* \subseteq S_1 \cup \dots \cup S_k$ and $t(S_k) + t_j \leq c$ (where E_j and L_j provide bounds on the earliest and latest possible station for task j, respectively).

Priority based methods apply priority rules (main rule and tie breaking rules) to rank tasks according to specific task weights (priorities). In each of n iterations a task with the highest priority is chosen from a set of available tasks and assigned to a station. Two different strategies for constructing this set may be distinguished. In a *stationoriented procedure* (cf. Talbot et al. (1986)), starting with station 1 the stations are considered successively. In each iteration a task with highest priority which is assignable to the current station k is chosen and assigned. The current station is closed and the next station $k + 1$ is opened when no more task is assignable to k. In contrast, within a *taskoriented procedure* (cf. Hackman et al. (1989)) among all available tasks one with highest priority is chosen and assigned to the earliest station to which it is assignable.

In the sequel we restrict ourselves to the application and discussion of stationoriented procedures. This is due to the results of Scholl and Voß (1994) who compared various priority rules (including the ones discussed below) with respect to the concept of both strategies and concluded a superiority of stationoriented procedures over taskoriented ones.

In the literature a great variety of **priority rules** has been proposed and compared (see, e.g., the comprehensive investigations of Talbot et al. (1986) and Scholl and Voß (1994)). A selection of rules which has been proven successful elsewhere is presented here. Tasks may be ranked according to:

• descending task times t_j	(MaxTime)
• descending positional weights $pw_j := t_j + \sum_{h \in F_j^*} t_h$	(MaxPW)
• descending number of followers $\lvert F_j^* \rvert$	(MaxF)
• descending number of immediate followers $\lvert F_j \rvert$	(MaxIF)
• descending task time divided by latest station t_j / L_j	(MaxTimeL)
• descending task time divided by slack t_j / s_j	(MaxTimeSlack)

Following the results of Scholl and Voß (1994), the rules MaxTimeL, MaxTimeSlack (not included in former evaluations), MaxF, and MaxPW perform best among a large number of rules.

Within all rules, whenever an upper bound on the number of stations is used, it is initially set to $\bar{m} := \min\{n, \lfloor t_{sum}/(c + 1 - t_{\max}) \rfloor + 1, \lfloor 2 \cdot t_{sum}/(c + 1) \rfloor + 1\}$; see Hackman et al. (1989) for a slightly modified formula. Correspondingly, two simple lower bounds are considered within our calculations. The first bound is $\lceil t_{sum}/c \rceil$. The second one considers the number of tasks with their task time exceeding $c/2$ (because they have to be assigned to different stations) as well as those with $t_j = c/2$ two of which may share a station.

Note that stationoriented priority based procedures develop the solutions in ascending order of stations. In that sense such procedures may be referred to as working *unidirectionally* from station 1 to subsequent stations in a **forward** manner. An easy modification can be made to obtain **backward procedures** by reversing the precedence graph (reversing the direction of all arcs) and applying the respective algorithm to the resulting problem. In a figurative sense any problem might be 'mirrored' and then the backward rule becomes the forward rule and vice versa. The motivation for this modification lies in the results of some careful target analysis for a number of benchmark problems revealing a strong data dependency in some cases.

Furthermore, the procedures can be extended to assign tasks **bidirectionally**, i.e., to apply a forward and backward procedure simultaneously. This is achieved by building a forward and a backward ranking, respectively. In the backward case the priority rules have to be applied to the reversed problem. The same is true for the definitions of available and assignable tasks: A task is *backward available* if all successors (= predecessors in the reversed graph) are already assigned and *backward assignable* to station k if it is backward available and no successor is assigned to an earlier station.

Bidirectional procedure: It starts with stations k_f and k_b as current stations where $k_f = 1$ and k_b is equal to a sufficiently large number (e.g., n). In each iteration one task with highest priority which is either (forward) assignable to k_f or backward assignable to k_b is chosen and assigned to the respective station. If no further task is assignable to $k_f(k_b)$ the next station $k_f + 1(k_b - 1)$ is considered. The algorithm stops when all tasks are assigned. A feasible solution is obtained by combining the partial forward and backward solutions.

In case that an upper bound $\bar{m}$ on the number of stations is known the procedure may be modified to search only for solutions with at most $\bar{m} - 1$ stations. Then k_b may be initially set to $\bar{m} - 1$ and a second stopping criterion is to stop whenever $k_b < k_f$ holds. Stopping due to this criterion means that no feasible solution with less than $\bar{m}$ stations can be found with the present rule.

For tasks which are simultaneously in the set of forward and backward available tasks an additional priority rule may be applied to decide whether they are assigned to their earliest or latest station. (Here even a rule referring to the actual station loads may be helpful as, e.g., preferring that station whose station time gets closer to c).

Example: Fig. 1 shows the precedence graph of a problem with $n = 10$ tasks. We assume that the cycle time is $c = 14$ and that a solution with $\bar{m} = 5$ stations is already known. The problem is solved by applying a forward, a backward and a bidirectional procedure using the rule MaxF with MaxTime and original index as tie breakers. The resulting task ordering is $\langle 1, 2, 5, 3, 4, 8, 6, 7, 9, 10 \rangle$ in the forward case and $\langle 10, 9, 8, 5, 4, 3, 6, 7, 2, 1 \rangle$ in the backward case.

Forward procedure: The resulting solution is $S_1 = \{1, 2, 5\}$, $S_2 = \{3, 4, 8, 6\}$, $S_3 = \{7, 9\}$, $S_4 = \{10\}$ with $m = 4$ stations.

Backward procedure: Task 10 is the only backward available task. After assigning it to station 1, tasks 6, 7 and 9 are backward available and task 9 has the highest priority. It is assigned to station 1 as well as tasks 8 and 5. The remaining assignments are easily determined and the corresponding forward solution is $S_1 = \{1\}$, $S_2 = \{7\}$, $S_3 = \{4, 3, 6, 2\}$, $S_4 = \{10, 9, 8, 5\}$ with $m = 4$.

Bidirectional procedure: The solution is obtained through the following assignments.

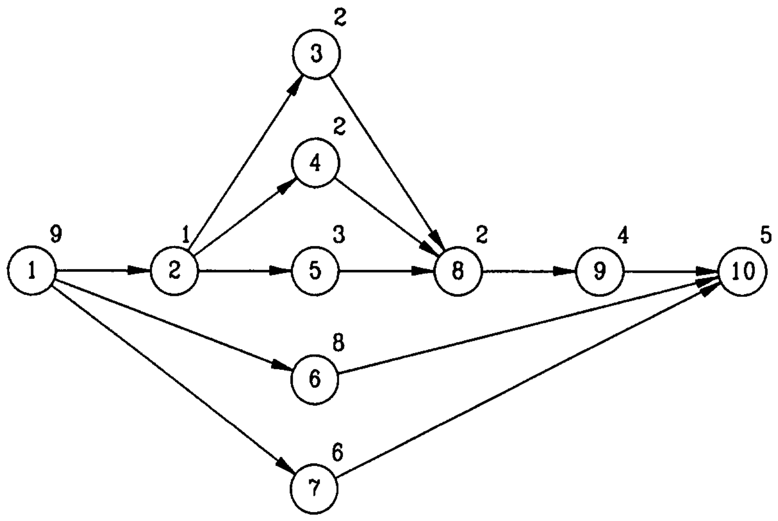

Figure 1.

Task 1 is forward and task 10 backward available. Due to $|F_1^*| = |P_{10}^*|$ (note the priority correspondence due to the precedence graph reversion) and $t_1 > t_{10}$ task 1 is chosen and assigned to station 1. Now, tasks 2, 6, and 7 are forward available and task 10, which has the higher priority, is backward available (assign 10 to station $\bar{m} - 1 = 4$). Tasks 6, 7, and 9 become backward available. Due to $|F_2^*| = |P_9^*|$ and $t_9 > t_2$ task 9 is assigned to station 4. With the remaining assignments the solution $S_1 = \{1, 2, 3, 4\}$, $S_2 = \{6, 7\}$, $S_3 = \{5, 8, 9, 10\}$ with $m = 3$ stations is obtained.

Dynamic rules: All rules considered above are of *static* nature since priorities only depend on parameters not being influenced by the solution process. *Dynamic* rules can be obtained by using modified parameters. The values of $E_j(L_j)$ may be modified by computing the earliest (latest) station in which a task j is *currently* assignable. In case of bidirectional planning the number or sum of task times of unassigned successors or predecessors may be used instead of all successors or predecessors. Therefore, we may define dynamic counterparts for all rules with the exception of MaxTime.

Even task-independent parameters like, e.g., the numbers of currently forward and backward available tasks, respectively, may be included in bidirectional rules to help deciding whether a forward or a backward step has to be performed. Following the idea that the probability of making a disadvantageous assignment decreases with decreasing number of available tasks our rule MaxTimeSlack may be extended to a dynamic bidirectional rule MaxTSAvail:

> Rank tasks according to descending values of t_j/s_j divided by the number of forward and backward available tasks, respectively. Tasks which are simultaneously forward and backward available may get different priorities for both directions.

To the best of our knowledge this rule belongs to a new class of priority based methods because it is especially devised for the dynamic bidirectional planning concept.

Considering the extensive discussion of priority rules within other problem domains this is an innovative idea with potential applications in various fields of operations management including precedence related activities. For instance, advising such procedure in the context of the bowl phenomenon (see, e.g., Hillier and So (1993)) should provide an interesting new avenue of research. This is due to the tendency of bidirectional procedures to concentrate idle time in the middle of the line instead of accumulating it at the end (or start) like forward (backward) procedures do.

2.2. *Search Methods for SALBP-2*

Most approaches for SALBP-2 use a search method which solves SALBP-1 instances for several cycle times to determine the smallest cycle time for which a feasible assignment to m stations exists (*feasibility problem*). If all these SALBP-1 instances are solved to optimality the SALBP-2 solution is optimal, otherwise it may be suboptimal. Such methods start with an interval $[LB, UB]$ of possible cycle times. The lower bound can be computed, e.g., by $LB = \max\{\lceil t_{sum}/m \rceil, t_{\max}\}$. In addition, this bound may be strengthened by determining the smallest cycle time such that for all tasks j the inequality $E_j \leq L_j$ holds (provided that $\bar{m} = m + 1$ is used in the calculation of L_j).

An upper bound UB may result from a minimum production rate or a known heuristic solution. Here it is calculated as follows (cf. Hackman et al. (1989)):

$$UB := \max\{t_{\max}, \left\{ \begin{array}{ll} \lfloor 2 \cdot (\Sigma t_j - 1)/m \rfloor & m \text{ even} \\ \lfloor 2 \cdot \Sigma t_j/(m+1) \rfloor & m \text{ odd} \end{array} \right\}\}$$

Two general search strategies are (see, e.g., Dar-El (1975), Wee and Magazine (1981), and Hackman et al. (1989)):

- **Lower bound method**: It starts with $c = LB$ and uses a solution procedure for SALBP-1 to check whether an assignment of all tasks to m stations exists. Either a solution is found and the algorithm terminates, or c is increased by 1 and the process is repeated.

- **Binary Search**: In each iteration the interval $[LB, UB]$ is bisected by choosing the cycle time c-$\lfloor (LB + UB)/2 \rfloor$. If for a given c a feasible solution with at most m stations is found the upper bound UB is set to $\max\{t(S_k) \mid k = 1, \ldots, m\}$. Otherwise LB is set to $c + 1$. The search stops when $LB = UB$.

These elementary methods can be modified and combined. Since a lower bound method may have to solve a large number of SALBP-1 instances if the cycle time of the optimal solution of SALBP-2 is much larger than the initial LB other increments for c, e.g., the Fibonacci numbers $1, 2, 3, 5, \ldots$ can be used. In this case the trial cycle times are LB, $LB + 1$, $LB + 3$, $LB + 6$, $LB + 11$, ... (Fibonacci search). When a new upper bound is determined a binary search or a lower bound method with smaller increments can be applied to the remaining interval.

An upper bound method (i.e., starting with UB and successively decreasing by 1) is not considered because it performs poorly if the optimal solution is close to a lower bound leaving a remarkable gap between this bound and the probably large value of UB which is the case for most problem instances of all three data sets considered below (cf. Section 4.1).

3. The Tabu Search Procedure

Tabu search is a metastrategy for guiding heuristic improvement procedures to overcome local optimality. For a background on tabu search see Glover (1989, 1990), Glover and Laguna (1993) and Voß (1993). Successful applications of this metaheuristic to problems somewhat related to SALBP are reported by, e.g., Hübscher and Glover (1994) as well as Voß (1994).

Within the basic concept of tabu search we try to improve a given feasible solution by *transforming* it iteratively into other feasible solutions while allowing for intermediate deteriorations of the objective function values. Transformations are referred to as moves and may be described by a set of one or more *attributes.*

The most important component of tabu search is the *tabu list management.* It concerns updating the so-called *tabu list,* i.e., deciding on how many and which attributes have to be set tabu within any iteration of the search. We may distinguish two main approaches: static methods and dynamic methods. In the sequel we focus on a static concept, which up to now seems to be the most popular one that has been applied in the literature. For dynamic concepts we refer to Voß (1993).

Trying to apply tabu search to SALBP-1 reveals the general problem of naturally defined shift and swap moves in the presence of hard capacity restrictions. This definition does not allow for a clear distinction between different moves with respect to their solution quality because the objective function value only changes with the number of stations being modified. Therefore, a diverting strategy has to be developed which, indeed, is a concept of general interest for tabu search applications.

In the sequel an improvement procedure for SALBP-2 is developed and combined with tabu search. Some additional strategies for, e.g., modifying the basic tabu list management are investigated. Finally, the application of tabu search to SALBP-1 is discussed following a general idea of overcoming the boundaries of feasibility.

3.1. A Tabu Search Approach for SALBP-2

A basic improvement procedure for SALBP-2 starts from a feasible solution and may consist of so-called shifts and swaps which can be explained by use of the following notation:

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

The following neighbourhood definition may be used.

A *shift move* (j, k_1, k_2) describes the movement of a task j from station k_1 to station k_2 in case of $LP_j \leq k_2 \leq ES_j$. Two shifts (j_1, k_1, k_2) and (j_2, k_2, k_1) build a *swap move* exchanging tasks j_1 and j_2, which are not related by precedence, between different stations k_1 and k_2. Swaps are abbreviated by (j_1, k_1, j_2, k_2).

A station k is called *critical* if $t(S_k) = \max\{t(S_i) \mid i = 1, \dots, m\}$ holds, i.e., station k has no idle time and determines the cycle time c which is to be minimized.

In a **steepest descent approach** in each iteration a critical station k_1 is chosen and a shift or a swap containing k_1 and a non-critical station k_2 is performed which results in the largest reduction of the cycle time. If a shift is interpreted as a swap with a fictive task j_2 of station k_2 with zero task time the new value $\bar{c}$ of the cycle time is determined by

$$\bar{c} = \max\{t(S_{k_2}) + t_{j_1} - t_{j_2}, t(S_{k_1}) + t_{j_2} - t_{j_1}, \max\{t(S_k) \mid k \neq k_1 \text{ and } k \neq k_2\}\}.$$

Since more than one station may be critical, moves which do not change the current cycle time must be allowed, too. Then, however, a serious problem of cycling within the search space may occur and corresponding methods to prevent this phenomenon have to be developed. Otherwise, the process stops if only shifts or swaps exist which would increase the current cycle time.

A similar approach may be used for the related problem of improving the equality of work loads (for this problem see Moodie and Young (1965), Rachamadugu and Talbot (1991)).

To overcome local optimality the above approach may be extended to a **steepest descent/mildest ascent approach** by always choosing a move (shift or swap) which leads to the lowest possible value of the new cycle time. If no improving move exists one with a least increase of the cycle time is performed. In many cases this approach leads to cyclic re-visiting of already encountered solutions since precedence relations may cause very restricted neighbourhoods for particular solutions. To avoid this cycling, static tabu search as a metastrategy to control the solution process may be applied as follows.

Static Tabu Search—Basic Approach

In our approach single attributes are set tabu as soon as their complements have been part of a selected move. A shift (j, k_1, k_2) may be described by the attributes "task j leaves station k_1" and "task j enters station k_2" which are denoted by $(j, \bar{k}_1)$ and (j, k_2), respectively. To avoid reversing the move, the attribute (j, k_1) which is complementary to $(j, \bar{k}_1)$ is set tabu. Since swaps are composed of two shifts, they are characterized by the attributes $(j_1, \bar{k}_1)$, (j_1, k_2), $(j_2, \bar{k}_2)$, and (j_2, k_1). The attributes (j_1, k_1) and (j_2, k_2) are set tabu by writing them into a cyclic tabu list of given *length TL* following the FIFO-strategy. Moves containing one or two of the attributes which are in the tabu list are forbidden (tabu). So, the algorithm selects in each iteration an (admissible) move which is not tabu and leads to the best possible cycle time. The procedure may be stopped due to one or more of the following criteria:

- a given number of iterations (MaxIt) has been performed
- a prespecified time limit has been reached
- a given number of iterations has been performed without improving the best known solution

- the solution has been proved to be optimal since its cycle time is equal to a known lower bound

If in an iteration no admissible move is available a so-called conflict management strategy is followed, i.e., the process may be continued by

- successively deleting the oldest moves from the tabu list until the list is empty or an admissible move exists, or by
- shifting predecessors (successors) of tasks of critical stations to earlier (later) stations to enlarge the neighbourhood.

Modifications of the Basic Approach

Although quite efficient at times, the basic idea seems to be somehow limited. As is the case with our approach to SALBP-2 the quality of solutions often strongly depends on the chosen value of TL since a long tabu list may exclude more and a short tabu list less moves than necessary. Therefore, some efforts have been made to include dynamic aspects like, e.g., random variation of TL within a specified interval. Another approach may be to activate or deactivate parts of the tabu list randomly or periodically.

Some additional procedures may improve the basic approach:

- **Global aspiration criterion**: For the sake of an improved effectivity an *aspiration level criterion* permits the choice of an attribute or move even when it is tabu. This is advantageous when a new best solution may be calculated, or when the tabu status of the attributes prevent any move from feasibility.
- **Varying *TL***: The choice of an appropriate length of the tabu list seems to be very sensitive with respect to solution quality. Therefore, TL may be changed after a certain number of iterations without improving the best known solution. This can be done randomly or systematically, e.g., the size of the list may be successively reduced by a constant length.
- **Dynamic parts of the tabu list (moving gap)**: One may use a tabu list which is subdivided into a static and a dynamic sublist. While the static sublist is always active only a part of the dynamic sublist is used at a time (see, e.g., Hübscher and Glover (1994)).
- **Intensification and diversification strategies**: Intensification aims at concentrating the search to a specific region of the solution space whereas diversification tries to lead the search trajectory into yet unvisited regions of the solution space. These strategies can be accomplished by means of a so-called *frequency based memory*.

3.2. *Application of Tabu Search to SALBP-1*

With respect to applying tabu search to SALBP-1 devising a suitable and efficient neighbourhood is not obvious. For this problem the production rate imposes very restrictive capacity constraints and the usual definition of shift and swap moves hardly has any effect on the objective function value.

That is, assume a given feasible solution with objective function value m, a reduction of m is only possible whenever all tasks of a station are removed and assigned to other stations; either there is at least one station with exactly one task or no improving move exists. Hence, all possible moves may be grouped into three classes, the first consisting of improving moves (i.e., m reduces by 1), the second of moves with no change of m, and the third of those where m deteriorates by 1. Within these groups no further obvious distinction of moves is possible. When m gets closer to optimality the available idle time of the line reduces and the size of the neighbourhood (regarding feasible moves) reduces, too. Then, taking into account the precedence constraints, in most iterations the first class of moves is empty and one has to choose among the moves of the second class or even the third one without having a respective distinction or ranking measure within a class.

Therefore, directly applying tabu search to SALBP-1 seems to be very difficult because there is no natural measure to find 'good' moves. The overall best objective function value may change, eventually, only after a large number of iterations when one station with zero station time can be achieved. In such a case strategic oscillation may come into play, but again the same difficulties tend to occur.

Due to these problems one may utilize the relationship between SALBP-1 and SALBP-2, which in some sense may be referred to as dual problems, in a rather simple way. Based on the relationship between both problem types the tabu search procedure proposed for SALBP-2 may be applied to SALBP-1 in the framework of a general search method.

For several values m of the number of stations the tabu search algorithm is used to determine a feasible solution with a cycle time not larger than the given one. If such a solution is found, m is an upper bound on the number of stations, otherwise the search continues with a larger value of m. All methods described in Section 2.2 may be used in principle, but the lower bound method seems to be superior because most problem instances of available data sets from the literature (see below) have optimal numbers of stations which are close to the trivial lower bound $LB = \lceil t_{sum}/c \rceil$. In the framework of the lower bound method m is initially set to LB and the tabu search procedure is applied for a certain number of iterations. If a feasible solution is found, the search stops, otherwise m is increased by one and the process continues. For an additional stopping criterion an upper bound may be calculated before applying the lower bound strategy.

The tabu search approach is not strategically oscillating between feasibility and infeasibility, but it is diverted to another problem having the same feasibility problem. That is, the search is started within the infeasible region and 'moved' towards feasibility. Whenever a feasible solution with $m > LB$ is found the approach might be modified by reducing m again to search for a solution with a better objective function value (eventually with a different parameter setting of the search method).

This strategy is a general purpose approach applicable where devising a suitable neighbourhood is not obvious. To be more specific, such an approach may be helpful in cases where related optimization problems have to be considered which have the same underlying decision problem (cf. the term feasibility problem above), as is the case with SALBP-1 and SALBP-2. For example, the same relationship exists between the bin packing problem and the problem of scheduling identical parallel machines with the objective of minimizing the makespan.

4. Computational Experiments

In the sequel we report on computational experiences with the heuristic approaches presented above. In Section 4.1 we describe the used data sets as well as the hardware and software environment of the tests. In Section 4.2 we compare unidirectional and bidirectional priority based procedures for SALBP-1, using static as well as dynamic rules, and their iterative application to solve SALBP-2. Sections 4.3 and 4.4 contain results of tests with tabu search procedures for SALBP-2 as well as for SALBP-1.

4.1. Data Sets and Experimental Conditions

In the last four decades a number of papers included real world as well as fictive data for SALBP. Some problems with 8 to 111 tasks were collected by Talbot et al. (1986) to form a data set for SALBP-1 with 64 instances which has been extensively used as benchmark up to now. Since most of these instances are easy to solve, Hoffmann (1992) proposed a procedure to obtain "harder" cycle times for the same set of problems. This results in a set of 50 instances 13 of which are in the set of Talbot et al., too. We collected additional problems with up to 297 tasks and applied the same procedure to generate a new data set for SALBP-1 with 168 instances which seems to be more challenging than the former ones. The complete problem definitions and solutions are given in Scholl (1993). For our tests these three data sets are combined to one set with 269 instances 256 of which are presently solved to optimality. (Note that our combined data set does not include those instances twice which are in both data sets of Talbot et al. and Hoffmann.)

For SALBP-2 no benchmark data set is available in the open literature. Therefore, we propose such a data set with 302 instances which contains the same problems as the data sets for SALBP-1. Problem instances are built by defining several feasible intervals of consecutive values for m, the given number of stations. For 265 of these instances the optimal solution is actually known. Both the instance definitions as well as the optimal, or best known cycle times, respectively, are documented in Scholl (1993).

All tested procedures are programmed using Borland's Pascal 7.0. The tests are performed on an IBM-compatible personal computer with an 80486 DX2-66 central processing unit.

4.2. Priority Based Methods

In this section we report on computational experiences with different versions of priority based methods. When computation times are given they refer to the times excluding input and output calculations as well as the computation of the transitive closure.

4.2.1. Results for SALBP-1

The first test attempts to compare unidirectional and bidirectional strategies. Following the results of Scholl and Voß (1994) which are in accordance with the studies of Talbot et al. (1986) and Hackman et al. (1989) who compare a large number of priority rules the four static rules MaxTimeL, MaxTimeSlack, MaxF, and MaxPW are chosen due to their best behaviour compared to other rules. Here they are applied in a forward and a backward procedure, respectively. Furthermore, these and the rule MaxTSAvail are used as dynamic rules in bidirectional procedure. For MaxF the rule MaxTime is used as tie breaker whereas for all other rules MaxIF serves as tie breaker. As second level tie breaker we use the original index.

Table 1 shows the summarized results for these 13 rules (for our combined data set with 269 instances), containing the following information together with average and maximal computation times (given in seconds).

- # opt.: number of instances for which an optimal solution is found[2]
- # best: number of instances for which the respective rule provides the best solution found by any of the rules applied in the corresponding test
- av. dev. (%): average relative deviation from optimal solution in % [2]
- max. dev. (%): maximum relative deviation from optimal solution in % [2]
- av. dev. (abs): average absolute deviation (additional stations) from optimum [2]
- max. dev. (abs): maximum absolute deviation (additional stations) from optimum [2]

With respect to average relative deviation from optimality and the number of optimal solutions the best rules are Bi-MaxTSAvail, F-MaxTimeL, Bi-MaxTimeSlack, Bi-MaxF and Bi-MaxPW (F = forward, B = backward, Bi = bidirectional). In all cases but MaxTimeL the bidirectional rules perform better than the respective unidirectional rules. In case of MaxTimeL the forward procedure performs best, but B-MaxTimeL is the worst rule in the test pointing at MaxTimeL being sensitive with respect to data changes. Since the average deviation of the bidirectional version of the rule is much better than the mean value of the deviations of both unidirectional rules, it may be the better choice.

The latter discussion also gives a clear indication that our bidirectional procedure substantially differs from a composite heuristic simply taking the best from the forward and the backward solutions, respectively. Instead, our approach combines partial solutions obtained from both directions simultaneously.

The average deviations in numbers of stations are almost equal for most of the rules, only the backward versions of MaxTimeL and MaxTimeSlack show significantly larger values. The maximal deviations are noteworthy (in three cases six additional stations and up to 50% in two cases). With respect to the number of best solutions the best rules are based on MaxF. Considering this measure it becomes apparent that very often several rules produce solutions with identical objective function values.

Table 1.

	Forward				Backward				Bidirectional				
Rule	MaxF	Max PW	Max TimeL	Max Time Slack	MaxF	Max PW	Max TimeL	Max Time Slack	MaxF	Max PW	Max TimeL	Max Time Slack	Max TS Avail
# opt.	129	132	138	134	131	124	120	121	133	134	132	136	138
# best	201	199	204	198	207	196	170	173	213	204	190	194	197
av. dev. (%)	3.87	4.02	3.63	3.78	3.75	3.82	4.31	4.16	3.70	3.67	3.84	3.58	3.25
max. dev. (%)	33.33	50.00	33.33	33.33	33.33	33.33	33.33	33.33	50.00	33.33	33.33	25.00	25.00
av. dev. (abs)	0.71	0.72	0.70	0.72	0.68	0.72	0.89	0.88	0.67	0.72	0.80	0.76	0.71
max. dev. (abs)	6	6	3	3	4	4	5	4	5	6	4	4	3
av. cpu time	0.03	0.03	0.03	0.03	0.02	0.04	0.04	0.03	0.16	0.15	0.13	0.14	0.15
max. cpu time	0.28	0.28	0.22	0.28	0.27	0.27	0.33	0.28	0.88	0.88	0.88	0.83	0.83

The results may be summarized as follows. Forward rules may outperform backward rules and vice versa showing their data dependency since backward procedures act as forward procedures on reversed precedence graphs. This disadvantage is removed by bidirectional rules which are able to choose the planning direction by dynamically evaluating the structure of the precedence graph.

Whereas the bidirectional procedures perform well with respect to solution quality the computation times are much larger than those for unidirectional procedures. An average factor of five holds. Since all times are smaller than 1 cpu second, this additional effort seems to be negligible.

Let us note that in case of unidirectional procedures rules incorporating slack values may be used as dynamic rules by computing dynamic values of E_j in each iteration of the algorithm. The application of these dynamic rules to our data sets results in very small reductions of the average deviations ($\leq 0.1\%$ for the rules given in Section 2.1) and increased computation times (about three times larger than those for static rules). Therefore, the gain of dynamic rules seems to be negligible in case of unidirectional procedures.

To conclude, our tests have shown that new rules as MaxTimeSlack and bidirectional planning including MaxTSAvail may help to improve the quality of solutions for SALBP-1. Consequently, these rules may also be helpful in solving SALBP-2 which is evaluated in the next section.

4.2.2. Results for SALBP-2

In this section we compare the influence of priority rules for SALBP-1 and of various search methods for solving SALBP-2. We perform the tests on the new data set with 302 instances (see Section 4.1). First, we apply the binary search method and compare the 13 rules of the test in Section 4.2.1. The results are summarized in Table 2.

It can be stated that all rules perform well due to their average relative deviation values not exceeding 3.28%. A random rule as another reference procedure results in an average (maximal) deviation value of 5.6% (18%). Despite the small differences between the rules some interesting observations can be made. MaxF and MaxPW outperform the other rules with respect to average deviation, especially in the bidirectional case. Both rules prefer tasks with many successors which results in an enlargement of the degree of freedom for choosing tasks in subsequent iterations. This may be especially important in a binary search procedure for SALBP-2 because only feasible solutions have to be found for combinations of c and m instead of an optimal one in case of SALBP-1. Although the rules MaxTimeL and MaxTimeSlack show larger average deviations they find more optimal solutions underlining their efficiency in solving SALBP-1. The new rule Bi-MaxTSAvail may be a good compromise considering both mentioned aspects because it outperforms MaxF and MaxPW with respect to the number of optimal solutions and MaxTimeL and MaxTimeSlack with respect to average solution quality. Similar to SALBP-1 the bidirectional rules need computation times which are up to six times larger than those for unidirectional rules.

The next test is performed to investigate the impact of different search methods. It comprises the lower bound method (LBM), binary search (BS), combined Fibonacci and binary search (FBS) and binary search with prespecified entry point (EBS).

Table 2.

	Forward				Backward				Bidirectional				
Rule	MaxF	Max PW	Max TimeL	Max Time Slack	MaxF	Max PW	Max TimeL	Max Time Slack	MaxF	Max PW	Max TimeL	Max Time Slack	Max TS Avail
# opt.	22	22	31	28	25	25	29	29	28	25	34	31	33
# best	90	71	86	81	92	73	67	69	117	92	90	84	93
av. dev. (%)	2.57	2.57	2.74	2.79	2.47	2.52	3.28	3.25	2.29	2.58	2.78	2.76	2.63
max. dev. (%)	12.50	12.50	11.67	11.67	11.67	10.71	12.12	12.12	11.67	12.99	11.84	13.33	10.67
av. cpu time	0.26	0.25	0.23	0.23	0.25	0.25	0.26	0.26	1.43	1.42	1.23	1.26	1.36
max. cpu time	1.92	1.87	1.87	1.70	1.87	1.76	1.93	1.98	8.79	8.85	7.80	8.40	8.79

Table 3.

Search method	LBM	BS	FBS	EBS
# opt.	33	33	33	33
# best	301	252	259	248
av. dev. (%)	2.29	2.63	2.51	2.62
max. dev. (%)	9.26	10.67	10.67	10.67
av. # iter.	29.47	7.93	7.05	6.78
max. # iter.	286	15	19	14
av. cpu time	4.92	1.36	1.37	1.21
max. cpu time	43.50	8.78	10.55	9.01

LBM and BS are applied as described in Section 2.2, FBS consists of a Fibonacci search until the first feasible solution is found and a binary search on the remaining interval. EBS differs from BS by the choice of the first trial cycle time which is set to the largest value of c for which $m = \lceil t_{sum}/c \rceil$ holds (cf. Wee and Magazine (1981)). Based on our above results the bidirectional dynamic rule Bi-MaxTSAvail is used as priority rule.

The results for our data set are summarized in Table 3. Compared to previous tables it provides information on the average and the maximum number of iterations (av. and max. # iter.) of the search procedure, i.e., the number of trial cycle times.

LBM performs best with respect to solution quality but needs the largest computation times because the number of iterations is equal to the difference between the determined cycle time and the lower bound LB plus 1. The other methods successively subdivide their search interval. Among these, FBS is slightly dominating because it combines a lower bound method and a binary search. The results of EBS and BS are almost identical with a smaller average number of iterations for EBS since it chooses a first trial cycle time which is closer to the optimum especially for large search intervals.

Since FBS provides good results in short time, it should be the best choice to compute initial feasible solutions for improvement procedures.

4.3. Tabu Search for SALBP-2

In this section we report on computational experiments with tabu search procedures for SALBP-2. We use the same data set with 302 instances as in Section 4.2.2 and perform several experiments to evaluate various factors which may influence the performance of the tabu search procedure.

Various Sizes of Tabu List

The first test concerns the length TL of the tabu list. As proposed in many references on tabu search we use the value $TL = 7$ (abbreviated by "F7"). Furthermore, we use the fixed length $TL = 15$ (F15), because preliminary tests have indicated that this length is appropriate for the data set used. A third procedure (S15) modifies TL by using uniformly distributed random numbers as follows: TL is restricted to the interval $[15 - 7, 15 + 7]$. Initially, $TL = 15$ is used. After $\lfloor \text{MaxIt}/10 \rfloor$ iterations without improving the actual best solution the list length TL is decremented or incremented by 1 or keeps the same value with probability $1/3$ for each choice. If $TL = 8$ ($TL = 22$) the probability is $1/2$ for keeping the value or incrementing (decrementing) TL by 1.

A further method (DTL) incorporates some data dependency. It systematically decrements TL after a number of non-improving iterations. This number is determined by equally subdividing the remaining iterations among the remaining values of TL. Starting with $TL = \lfloor (m + n)/3 \rfloor$, the list length is successively reduced by 3 until it would become smaller than 3. For the actual data set the starting value of TL is between 12 for the smallest problem with 29 tasks and 7 stations and 116 for the largest one with 297 tasks and 52 stations.

To compute initial feasible solutions the search method FBS with the priority rule Bi-MaxF is applied since both turned out to be effective as shown by our results above. The tabu search procedures stop when the current best solution is equal to the lower bound, when MaxIt = 50,000 iterations have been performed, or after 500 cpu seconds, whatever occurs first. The global aspiration criterion is applied in all cases. If in an iteration no admissible move can be found, i.e., all moves are set tabu, a certain conflict management strategy, which moves predecessors of tasks in critical stations to earlier and successors to later stations, is included. If the conflict could not be resolved the oldest tabu attribute is freed from its tabu status. A more detailed description and evaluation of conflict management is given below.

The results are summarized in Table 4 containing the same information as former tables and additional information on:

• av. it. best	average iteration number in which the best solution is found
• av. # GA	average number of iterations in which a move is chosen due to the global aspiration criterion
• av. # confl.	average number of iterations in which all admissible moves are set tabu (conflict) and the conflict management strategy fails

The results show that all versions of the tabu search procedure are able to considerably improve the quality of the initial solutions which have an average deviation from optimality of 2.28%. However, a pure steepest descent approach without tabu search results in an average deviation value of 2.18%. In this case the mean number of improving iterations is smaller than 1, i.e., the algorithm does not perform any or stops after few iterations. This observation indicates that the initial feasible solutions are often close to a local sub-optimum in which the improvement procedures get stuck very soon. Therefore, it can be stated that the application of a steepest descent/mildest ascent approach with tabu search and any reasonable list size is successful compared to a pure steepest descent procedure.

Nevertheless, the results show that the tabu list size is a sensitive parameter for solving SALBP-2. For the considered data set F7 is quite poor whereas F15 results in significantly better average solution quality. Since there are some problems for which F7 is the better

Table 4.

List management	F7	F15	S15	DTL
# opt.	139	161	161	157
# best	207	254	255	236
av. dev. (%)	0.69	0.43	0.41	0.46
max. dev. (%)	7.94	5.26	5.26	5.26
av. it. best	2743.48	6767.43	7042.85	11174.17
av. # GA	1.09	1.87	1.84	2.51
av. # confl.	9.72	148.71	165.24	59.65
av. cpu time	92.02	77.67	78.40	84.05
max. cpu time	500.05	350.15	347.07	395.14

choice, it seems to be promising to take varying list sizes as is the case with S15 and DTL.

Considering the average iteration of the last improvement (av. it. best) shows that F7 stops finding improved solutions much earlier than the other methods. Especially, DTL is able to find improved solutions in the late iterations. This seems to be due to the large list sizes at the beginning.

The application of the global aspiration criterion is successful for all procedures. Whereas it leads to an average of one move which improves the current best solution in case of F7 it enables DTL to find an average of 2.5 improvements.

Naturally, F7 produces the least number of iterations in which a conflict occurs that can not be solved by the conflict management strategy since the tabu list has only few entries. DTL reduces this value to one third of that of F15 and S15.

Besides showing the worst solution quality F7 takes the longest average computation times and exceeds the time limit of 500 seconds for four instances. This is due to the larger number of candidate moves in each iteration because only a few are set tabu. The longer times of DTL in comparison to F15 and S15 may rely on the shorter list sizes for small problems and in late iterations.

Moving Gap

Another idea concerning the tabu list management is referred to as moving gap, i.e., a part (gap) of the tabu list is periodically deactivated. In our application the tabu list is subdivided into a static part of length $\lfloor TL/3 \rfloor$ which is always active and a dynamic part containing the remaining list entries. The dynamic part itself is equally subdivided into four subparts two of which are deactivated at a time. This is done by cyclically moving (every $\lfloor \text{MaxIt}/20 \rfloor$ iterations) a window simultaneously covering two parts of the dynamic sublist.

This strategy is added to the four procedures compared above. The results are summarized in Table 5 showing small improvements of the solution quality for all methods.

Table 5.

List management	F7	F15	S15	DTL
# opt.	144	162	168	165
# best	202	256	266	247
av. dev. (%)	0.66	0.43	0.40	0.41
max. dev. (%)	7.94	5.26	5.26	5.26
av. it. best	6882.21	6960.03	7318.91	8239.97
av. # GA	0.75	1.45	1.45	2.24
av. # confl.	0.13	142.77	147.94	106.45
av. cpu time	108.99	87.72	84.90	85.85
max. cpu time	500.10	500.05	473.62	500.05

Table 6.

	Initial rule	Forward	Backward	Bidirectional	Random
Initial solution	av. dev. (%)	2.57	2.47	2.28	5.59
	max. dev. (%)	12.50	11.67	11.67	17.86
Steepest descent	av. dev. (%)	2.46	2.36	2.18	4.79
	max. dev. (%)	12.50	11.67	11.67	17.86
	av. it. best	0.49	0.68	0.64	2.46
Tabu search	# opt.	132	136	140	136
	# best	217	220	242	209
	av. dev. (%)	0.69	0.64	0.57	0.67
	max. dev. (%)	7.79	5.26	5.26	5.26
	av. it. best	1239.20	1325.24	1260.44	1283.66
	av. # GA	1.79	2.14	1.86	4.86
	av. # confl.	1.39	2.61	1.08	1.07
	av. cpu time	8.71	8.47	10.47	9.38
	max. cpu time	36.47	41.68	41.85	48.61

Influence of the Initial Feasible Solution

We examine the influence of the quality and the structure of different initial feasible solutions on the performance of our improvement procedures. Using FBS as search method the priority rule MaxF, which is shown to be successful in Section 4.2.2, is applied as forward, backward and bidirectional rule with tie breaker MaxTime (and original index as the second level tie breaker). As a reference procedure a bidirectional random rule is used, i.e., forward and backward priorities are determined by uniformly distributed random numbers.

At first, we examine the influence of the different initial solutions on a pure steepest descent procedure. As shown in Table 6 the random rule yields initial solutions with an average deviation which is about 3% larger than those of the other rules. In case of the latter rules the average deviations are reduced by about 0.1%, respectively, which is a rather poor result. For most instances the initial solutions can not be improved at all. Even the random solutions which are far from optimality can not be significantly improved. Consequently, a pure steepest descent procedure is not a suitable improvement procedure for SALBP-2 whatever initial solution is given.

The application of our tabu search procedure with DTL and moving gap strategy yields much better results. Already 1,000 iterations are enough to reduce average deviations to about 1% for all initial rules. The results for MaxIt $= 5,000$ are included in Table 6. The following observations are noteworthy:

- The influence of the initial solutions is rather limited since only small differences of solution qualities remain. For MaxIt $= 50,000$ the average deviations are reduced to values very close to 0.4% independent of the used initial rules.

- The bidirectional rule slightly outperforms the other rules. This may depend on the structure of solutions which are constructed bidirectionally. Since stations are maximally loaded before the next station is opened, a forward (backward) procedure tends to cumulate idle times in later (earlier) stations. By planning bidirectionally idle times are concentrated in the middle of the line. That may lead to an enlarged neighbourhood because it may be easier due to the precedence constraints to move tasks from early or late stations to stations in the middle than it is to remove them from these stations (cf. our hint at the bowl phenomenon in section 2.1).

- The results obtained for the case of initializing with the forward rule are dominated even by those when initializing with the backward or the random rule. This may be due to the fact that our procedure chooses among several best moves the one which is found first. That is, it concerns the earliest among the stations in question. Subsequently, that may result in a restricted set of neighbour solutions since the early stations are heavily loaded. In the backward case, this search concentrates on stations with more slack. The random rule may produce more equally loaded stations.

Conflict Management Strategy

As mentioned above, a strategy for managing conflicts (no admissible move is available) has to be included. The simplest approach (abbreviated by C1) is to remove the tabu status of the oldest entries in the tabu list until an admissible move exists. At least after releasing all moves from the list an admissible move is found. Note that this move may invert the last one which may result in cyclic re-visiting of solutions.

In the second strategy (C2) we do not define a move for tasks within a critical station but for their respective immediate predecessors or successors, i.e., a most improving/least deteriorating move concerning the latter set of tasks is performed. The rationale behind this (diversifying) approach is to enlarge the degree of freedom for those (critical) tasks being trapped within critical stations.

Table 7.

Strategy	C1	C2	C3
# opt.	160	164	165
av. dev. (%)	0.50	0.44	0.41
av. # confl.	7789.89	4239.60	106.45
av. cpu time	96.67	106.58	85.85

In some cases the previous strategy still maintains the same shortcoming because a chain of predecessors or successors may prevent any reasonable move concerning a critical station. Therefore, in a third procedure (C3) we consider any move which either shifts (not necessarily immediate) predecessors of critical tasks to earlier stations or (not necessarily immediate) successors to later stations. In this case, the criterion to evaluate a move is not primarily based on the objective function but on the closeness of a respective task to a critical station (measured in the difference of respective station numbers). Ties are broken in favour of smaller objective function values.

All previous tests have been performed while applying C3 as conflict management strategy due to its superior behaviour compared to strategies C1 and C2. In Table 7 we present some numerical results for the same test as reported within Table 5 (column DTL with moving gap strategy and MaxIt $= 50,000$ iterations) underlying this behaviour. Column C3 gives the same results as presented above while the number of conflicts that have to be resolved are considerably increased for C2 and C1. Moreover, the solution quality decreases, measured in the respective deviation values, and even the cpu times favour C3 due to the fact that fewer conflicts have to be resolved throughout the search.

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 we use a frequency based memory in which the relative number of iterations any task j belongs to any station k is stored (denoted as z_{jk}). Five phases each of which being active for $\lfloor \text{MaxIt}/5 \rfloor$ iterations are distinguished: In the first phase the information for the frequency based memory is collected. In the second phase an intensification is achieved by fixing tasks which have mainly been in a certain station up to the current iteration. More precisely, all tasks j with $z_{jk} \geq 0.75$ are fixed to station k for $z_{jk} \cdot \lfloor \text{MaxIt}/5 \rfloor$ iterations. The third phase, without any intensification and diversification, is used to update the frequency based memory again. A diversification is achieved in the fourth phase by forbidding tasks j with $z_{jk} \geq 0.75$ to enter station k until z_{jk} falls below the value of 0.75 or an unresolvable conflict occurs. In the final phase all task-station-combinations are free again.

Our computational experience, however, implies that this approach does not lead to improvements in solution quality or time requirements (relative to the previously considered versions of our tabu search procedure). Therefore, this simple strategy does not seem to be efficient. Especially, the schematic application of intensification and diversification is poor.

4.4. *Tabu Search for SALBP-1*

In this section we report on some numerical results with the iterative application of our tabu search procedure to SALBP-1. As described in Section 3.2 this approach applies the SALBP-2 tabu search procedure within the framework of a lower bound method. It starts with the lower bound described in Section 2.1. For each trial station number a priority based procedure for SALBP-2 with Bi-MaxF as priority rule is applied to compute an initial solution for the tabu search procedure. The tabu search procedure itself includes DTL and the moving gap strategy as described above.

An initial value of an upper bound on the number of stations is obtained in two different ways before starting the lower bound method. First, we use a priority based method for SALBP-1 with the same rule as applied for SALBP-2, i.e., Bi-MaxF (the combination of this procedure with our tabu search approach is called PrioTabu). Second, we compute initial solutions with the hybrid procedure EUREKA of Hoffmann (1992) which combines a branch and bound procedure and a heuristic. The enumerative part of EUREKA is successively applied in forward and backward direction observing a given time limit, respectively. In case that no optimal solution has been obtained the heuristic of Hoffmann (1963) is applied. As recommended by Hoffmann (1992) we give a time limit of 3 seconds for both directions of the enumerative part (combined with our tabu search approach we call this procedure EurTabu).

In the sequel we compare the results of our tabu search procedure with the best known algorithms for SALBP-1 at hand. The latter are the branch and bound procedure FABLE of Johnson (1988) and the above mentioned hybrid procedure EUREKA, however, without combining it with the tabu search approach. Both procedures have been re-implemented by the authors of this paper by means of Borland's Pascal. Besides evaluating our approach, this comparison adds to clarify the recent discussion of Hoffmann and Johnson about the quality of FABLE and EUREKA, respectively (cf. Hoffmann (1993) and Johnson (1993)). Note that a comparison of the original FORTRAN codes gives very similar results.

In order to compare the exact and heuristic approaches, respectively, we give a time limit of LIMIT = 50, 100 and 250 seconds to all procedures. When the time limit is reached without proving optimality the station number of the best known solution provides an upper bound. In case of EUREKA the available time is equally split up between the forward and the backward branch and bound procedure while the heuristic is applied without time restriction. Note that in most cases where the heuristic must be applied it consumes only a fraction of a second. Only for the largest problem instances with 297 tasks up to 15 seconds of computation time are necessary. Indeed, this raises the general question about evaluating heuristics and exact procedures for which a parameter must be set. However, the results below are reported under the time setting as mentioned above. Astonishingly, by giving EUREKA a reduced time limit of 15 seconds for the heuristic and equally splitting up the remaining time for the forward and the backward part only the average cpu time is affected, i.e. reduced. Therefore, respective results are not explicitly presented.

In our tabu search procedures a total of $100 \cdot$ LIMIT iterations are performed for each trial station number m before m is increased. Preliminary computational tests have indicated that

Table 8.

Method	FABLE	EUREKA	PrioTabu	EurTabu
LIMIT = 50				
#opt.	192	211	190	211
av. dev. (%)	1.14	0.85	1.16	0.71
max. dev. (%)	10.00	12.00	14.29	7.69
av. cpu time	19.65	23.46	12.15	10.84
LIMIT = 100				
#opt.	194	214	195	211
av. dev. (%)	1.11	0.80	0.98	0.69
max. dev. (%)	10.00	12.00	14.29	7.69
av. cpu time	38.23	44.35	22.07	16.71
LIMIT = 250				
#opt.	198	214	199	213
av. dev. (%)	1.02	0.80	0.92	0.66
max. dev. (%)	7.69	12.00	14.29	7.69
av. cpu time	91.50	103.63	51.53	34.22

this number of iterations depending on the given time limit (LIMIT) is a suitable parameter choice.

Table 8 summarizes the results of a comparison between FABLE, EUREKA, PrioTabu and EurTabu. The results show that considerable improvements of the initial solutions for SALBP-1 are achieved by the iterative application of our tabu search procedure for SALBP-2. Even for the time limit of 50 seconds the average deviation is reduced from 3.70% (cf. Table 1) to 1.16% for PrioTabu. Furthermore, based on the average deviation value of 1.00% of the initial solution provided by EUREKA within EurTabu we gain a reduction to 0.71% (for LIMIT = 50) or 0.66% (for LIMIT = 250), respectively. Considering the average number of trial station numbers necessary for tabu search within both procedures PrioTabu and EurTabu, the decision to take a lower bound method is reasonable because in most cases only one iteration has to be performed (less than 1.5 on average).

Furthermore, with respect to solution quality (average deviation values) our PrioTabu approach is competitive to the best known procedures FABLE and EUREKA within shorter average cpu times. Within the given time limit of 250 seconds, e.g., the priority based tabu search approach as well as FABLE and EUREKA reach average deviation values between 0.8 and about 1%.

With EurTabu we go even further and clearly outperform the other approaches with significantly smaller average computation times. For instance, for LIMIT = 250 the average deviation of 0.80% for EUREKA is reduced by 17.5% to 0.66% for EurTabu. Of course, our heuristic approaches are only able to prove optimality for those instances where the found solution equals the initial lower bound whereas the possibilities of FABLE and EUREKA are more elaborate in this respect.

With respect to the different data sets it seems to be remarkable that both tabu search approaches yield optimal solutions for all 64 problems of the data set of Talbot et al. (1986) even for LIMIT = 15 with an average cpu time of less than two seconds. Concerning the other data sets a strong data dependency of achieved solution qualities is revealed underlining the importance of discussing various complexity measures of assembly line balancing problems (compare the recent discussion of Hoffmann (1993) and Johnson (1993)). With respect to certain regions of different measures, we can even show a more versatile dominance of our tabu search approaches over FABLE and EUREKA (cf. the discussion in the Appendix below).

5. Summary and Conclusions

In this paper we report on heuristic approaches for solving Type 1 and Type 2 of the simple assembly line balancing problem focusing priority based methods for finding initial feasible solutions as well as improvement procedures. We propose new priority rules, an extension to dynamic rules and bidirectional planning as a new concept. Furthermore, we describe a basic improvement procedure which is based on shifting tasks between stations. This improvement procedure is combined with a static version of tabu search, a recent meta-strategy to overcome local optimality.

Comprehensive numerical investigations on known as well as new data sets show that properly chosen priority rules and bidirectional planning result in good initial solutions for SALBP-1 as well as SALBP-2. For SALBP-1 these results are at least as good and in many cases superior to currently known approaches. Additional considerable improvements are achieved by our tabu search approaches which are able to outperform exact algorithms, provided that they observe the same time limits. For SALBP-2 priority rule based heuristics and the tabu search approach behave very successful with a good overall solution quality (measured with respect to lower bounds because no competitive algorithms are available in the published literature).

Some modifications of the basic tabu search procedure like the global aspiration criterion and the moving gap strategy are successful. The list size is a parameter which significantly influences the solution quality of our tabu search approach. Varying list sizes seem to be promising but additional effort has to be made to find better strategies than the proposed ones. A conflict management strategy overcomes problems arising with precedence constraints.

Since tabu search turns out to be a powerful tool in solving simple assembly line balancing problems, further investigations should deal with, e.g., dynamic approaches as well as its application in the field of mixed model assembly line balancing problems and other related problems like the bin packing and the parallel machine scheduling problem, respectively.

Acknowledgments

We wish to thank Professor Thomas Hoffmann and Professor Roger Johnson for providing us with copies of their codes of EUREKA and FABLE, respectively.

Table 9.

Problem classes		FABLE	EUREKA	PrioTabu	EurTabu
$0.25 < O \leq 0.55$	# opt. (%)	77.12	84.26	75.89	81.91
(127 instances)	av.dev. (%)	0.88	0.50	0.83	0.51
$0.55 < O \leq 0.85$	# opt. (%)	70.41	75.38	72.53	76.77
(142 instances)	av.dev. (%)	1.14	1.07	1.01	0.78

Appendix

In this appendix we provide some additional information on the comparison of our tabu search approaches for SALBP-1 and the algorithms EUREKA and FABLE by analyzing the used data sets. Common measures for classifying problem instances are the *order strength* O (number of arcs in the transitive closure of the precedence graph divided by $n \cdot (n-1)/2$) and the normalized *interval of task times* $[t_{\min}/c, t_{\max}/c]$. With respect to the order strength we subdivide our combined data set into two subsets of problem instances. The first subset contains 127 problem instances with $O \in (0.25, 0.55]$, the second one 142 problem instances with $O \in (0.55, 0.85]$. Note that all medium and large scale instances (≥ 30 tasks) in the data sets of Talbot et al. and Hoffmann fall into the restricted interval $[0.40, 0.55]$ (cf. Johnson (1993)). Table 9 summarizes the results of the tested procedures for LIMIT = 250 with respect to this distinction. It shows that the tabu search procedures are successful especially for problems with high order strength. The influence of the precedence constraints is twofold. While they reduce the number of feasible solutions, they reduce the possibilities to jointly assign tasks into stations, too. Therefore, problem instances with high order strength tend to have solutions with the number of stations exceeding known lower bounds, i.e., more idle time is necessary on the line. While in these cases enumeration procedures like FABLE and EUREKA tend to search in limited areas of the solution space, tabu search provides an intrinsic diversification possibility, based especially on our conflict management strategy.

As second measure we consider the ratio $VI := t_{\max}/t_{\min}$ which is somewhat related to the normalized interval of task times but excludes the cycle times. A small value of this *time variability index* VI indicates that the task times vary only in a small range or that the minimum task time is remarkably large. Therefore, respective problem instances are expected to be hard in the sense that task combinations are more difficult to determine than in case of small and/or considerably varying task times. Table 10 gives three intervals for VI and shows corresponding results for LIMIT = 250. For the problem set with $VI \in [1, 15]$ our tabu search approaches significantly outperform the enumerative approaches, because the tendency to have tasks with similar operation times increases the possibilities of exchanging tasks.

Table 10.

Problem classes		FABLE	EUREKA	PrioTabu	EurTabu
$1 \leq VI \leq 15$	# opt. (%)	77.01	81.61	81.61	86.20
(87 instances)	av.dev. (%)	1.08	1.08	0.77	0.48
$15 < VI \leq 80$	# opt. (%)	93.90	98.78	96.34	97.56
(82 instances)	av.dev. (%)	0.27	0.05	0.34	0.17
$80 < VI \leq 568.9$	# opt. (%)	54.00	62.00	49.00	58.00
(100 instances)	av.dev. (%)	1.58	1.16	1.52	1.21

Notes

1. Note that tasks j with $E_j = L_j$ may be assigned to E_j before applying any rule (otherwise, e.g., the slack values may be increased by a small number ϵ to avoid dividing by zero in the rule MaxTimeSlack below).
2. Whenever an optimal solution is not yet known or proven we refer to the value of the largest known lower bound as minimum number of stations.

References

Baybars, I. (1986). "A Survey of Exact Algorithms for the Simple Assembly Line Balancing Problem," *Management Sci.* 32, 909–932.

Charlton, J. M., and C. C. Death. (1969). "A General Method for Machine Scheduling," *Internat. J. Production Res.* 7, 207–217.

Dar-El, E. M. (1975). "Solving Large Single-Model Assembly Line Balancing Problems—A Comparative Study," *AIIE Trans.* 7, 302–310.

Domschke, W., A. Scholl, and S. Voß. (1993). *Produktionsplanung—Ablauforganisatorische Aspekte.* Berlin: Springer.

Glover, F. (1989). "Tabu Search—Part I," *ORSA Journal on Computing* 1, 190–206.

Glover, F. (1990). "Tabu Search—Part II," *ORSA Journal on Computing* 2, 4–32.

Glover, F., and M. Laguna. (1993). "Tabu Search." In C. R. Reeves (ed.), *Modern Heuristic Techniques for Combinatorial Problems*, pp. 70–150. Oxford: Blackwell.

Hackman, S. T., M. J. Magazine, and T. S. Wee. (1989). "Fast, Effective Algorithms for Simple Assembly Line Balancing Problems," *Oper. Res.* 37, 916–924.

Hillier, F. S., and K. C. So. (1993). "Some Data for Applying the Bowl Phenomenon to Large Production Line Systems," *Internat. J. Production Res.* 31, 811–822.

Hoffmann, T. R. (1963). "Assembly Line Balancing with a Precedence Matrix," *Management Sci.* 9, 551–562.

Hoffmann, T. R. (1992). "EUREKA: A Hybrid System for Assembly Line Balancing," *Management Sci.* 38, 39–47.

Hoffmann, T. R. (1993). "Response to Note on Microcomputer Performance of "FABLE" on Hoffmann's Data Sets," *Management Sci.* 39, 1192–1193.

Hübscher, R., and F. Glover. (1994). "Applying Tabu Search with Influential Diversification to Multiprocessor Scheduling," *Computers Ops. Res.* 21, 877–884.

Johnson, R. V. (1988). "Optimally Balancing Large Assembly Lines with "FABLE"," *Management Sci.* 34, 240–253.

Johnson, R. V. (1993). "Note: Microcomputer Performance of "FABLE" on Hoffmann's Data Sets," *Management Sci.* 39, 1190–1193.

Klein, R., and A. Scholl. (1996). "Maximizing the Production Rate in Simple Assembly Line Balancing—A Branch and Bound Procedure," *Europ. J. Oper. Res.* 91, 367–385.

Moodie, C. L., and H. H. Young. (1965). "A Heuristic Method of Assembly Line Balancing for Assumptions of Constant or Variable Work Element Times," *J. Industrial Engineering* 16, 23–29.

Rachamadugu, R., and B. Talbot. (1991). "Improving the Equality of Workload Assignments in Assembly Lines," *Internat. J. Production Res.* 29, 619–633.

Scholl, A. (1993). "Data of Assembly Line Balancing Problems," Working Paper, TH Darmstadt.

Scholl, A. (1994). "Ein B&B-Verfahren zur Abstimmung von Fließbändern bei gegebener Stationsanzahl." In H. Dyckhoff, U. Derigs, M. Salomon and H. C. Tijms (eds.), *Operations Research Proceedings 1993*, pp. 175–181. Berlin: Springer.

Scholl, A., and S. Voß. (1994). "A Note on Fast, Effective Heuristics for Simply Assembly Line Balancing Problems," Working Paper, TH Darmstadt.

Talbot, F. B., J. H. Patterson, and W. V. Gehrlein. (1986). "A Comparative Evaluation of Heuristic Line Balancing Techniques," *Management Sci.* 32, 430–454.

Voß, S. (1993). "Intelligent Search," Manuscript, TH Darmstadt.

Voß, S. (1994). "Tabu Search in Manufacturing." In H. Dyckhoff, U. Derigs, M. Salomon and H. C. Tijms (eds.), *Operations Research Proceedings 1993*, pp. 183–194. Berlin: Springer.

Wee, T. S., and M. J. Magazine. (1981). "An Efficient Branch and Bound Algorithm for an Assembly Line Balancing Problem—Part II: Maximize the Production Rate," Working Paper No. 151, University of Waterloo.