Jena Research Papers in Business and Economics

# Optimally solving the Alternative Subgraphs Assembly Line Balancing Problem

*Armin Scholl, Nils Boysen, Malte Fliedner*

05/2008

*Jenaer Schriften zur Wirtschaftswissenschaft*

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

ISSN 1864-3108

**Publisher:**

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

www.jbe.uni-jena.de

**Editor:**

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

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

www.jbe.uni-jena.de

# Optimally solving the Alternative Subgraphs Assembly Line Balancing Problem

**Armin Scholl**

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

**Nils Boysen**

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

**Malte Fliedner**

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

## Abstract

Assembly line balancing problems (ALBP) consist of distributing the total workload for manufacturing any unit of the products to be assembled among the work stations along a manufacturing line as used in the automotive or the electronics industries. Usually, it is assumed that the production process is fixed, i.e., has been determined in a preceding planning step. However, this sequential planning approach is often suboptimal because the efficiency of the production process can not be evaluated definitely without knowing the distribution of work. Instead, both decisions should be taken simultaneously. This has led to the Alternative Subgraphs ALBP.

We give an alternative representation of the problem, formulate an improved mixed-integer program and propose a solution approach based on SALOME, an effective branch&bound procedure for the well-known Simple ALBP. Computational experiments indicate that the proposed procedure is successful in finding optimal solutions for small- and medium-sized problem instances and rather good heuristic solutions for large-scaled instances.

## Keywords

Assembly line balancing – Production process – Mass-production – Combinatorial optimization – Sequencing

## 1 Introduction

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

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

The total amount of work is divided into a set of *tasks* $V = \{1,...,n\}$ which build the nodes of a precedence graph. Performing a task i takes a *task time* (node weight) $t_i$; the total production time per product unit is $ts(V) = \sum_{i \in V} t_i$. Technological and organizational conditions require to observe *precedence relations* (i,j) between different tasks i and j, i.e., task i must be finished before j is started. Non-redundant precedence relations are represented as arcs in the *precedence graph.* To simplify the presentation, we assume that graph $G = (V, E, t)$ is acyclical and numbered topologically. For each task $i \in V$, the precedence relations can be given by the set $P_i$ of direct predecessors and the set $F_i$ of direct successors. Including indirect precedence relations, the sets $P_i^*$ and $F_i^*$ of all predecessors and successors in the transitive closure $E^*$, respectively, are defined.

An ALBP generally consists of finding a feasible *line balance*, i.e., an assignment of each task to a station such that the cycle time constraints, the precedence constraints and possible further restrictions are fulfilled. The set $S_k$ of tasks assigned to a station k constitutes its *station load*, the *station time* $t(S_k) = \sum_{i \in S_k} t_i$ must not exceed the cycle time 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.

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

(S-1) Mass-production of one homogeneous product.

(S-2) All tasks are processed in a predetermined mode (no processing alternatives exist).

(S-3) Paced line with a fixed common cycle time according to a desired output quantity.

(S-4) The line is considered to be serial with no feeder lines or parallel elements.

(S-5) The processing sequence of tasks is subject to precedence restrictions.

(S-6) Deterministic (and w.l.o.g. integral) task times.

(S-7) No assignment restrictions of tasks besides precedence constraints.

(S-8) A task cannot be split among two or more stations.

(S-9) All stations are equally equipped with respect to machines and workers.

The main parameters m and c can be given as a constant or considered as a variable defining several problem versions. The most relevant version referred to as SALBP-1 ([ | |m] in the classification scheme of Boysen et al. 2007) is to minimize m given c. This problem and the other versions are NP-hard. Recent surveys covering SALBP models and procedures are given by Erel and Sarin (1998), Scholl (1999, ch. 2, 4, 5), Rekiek et al. (2002) as well as Scholl and Becker (2006).

## 2 The Alternative Subgraphs Assembly Line Balancing Problem (ASALBP)

In practice, there is usually a great flexibility in performing the work for manufacturing a product such as a car or a washing machine. This might concern only a single operation, e.g., a part might be connected to the workpiece by glueing, screwing or rivetting causing different execution times. Yet this might also span complete subgraphs of the precedence graph, e.g., a major part like the cockpit of a car can be assembled piece-by-piece inside the car body or a complete cockpit module can be inserted in a single operation. These examples show that prespecifying the whole production process prior to balancing the line faces the risk of loss in efficiency.

The new problem ASALBP, introduced by Capacho and Pastor (2005), overcomes this drawback as it considers processing alternatives within the precedence graph such that selecting the processes and assigning tasks to stations is done simultaneously. Up to now, the problem has been defined and modelled in a restricted version (Capacho and Pastor 2005) and an extended version (Capacho and Pastor 2006, 2007). Furthermore, constructive heuristic methods have been developed and tested comprehensively by Capacho et al. (2006, 2007). A special case of ASALBP which considers sequence-dependent task times is discussed by Scholl et al. (2006).

Limited former research on connecting processing alternatives and line balancing is due to Pinto et al. (1983). Another line of research indirectly considers processing alternatives by equipping stations with different machinery and tools which have different abilities and speeds to perform the tasks (e.g., see Bukchin and Tzur 2000 and the survey in Becker and Scholl 2006, Section 5).

The ASALBP[1] extends SALBP by removing the assumption (S-2). Instead, it is assumed that alternatives exist for parts of the production process. Each such part is called a *variable subprocess* which results in *alternative (disjunctive) subgraphs* in the precedence graph of which only one has to be chosen, respectively.

Capacho and Pastor (2005, 2007) propose the usage of a so-called S-graph to illustrate the alternatives by conventional arcs with a bracket indicating that only one of the arcs must be followed. We propose a more formal and established presentation which clearly shows the uniqueness of choosing a single out of several subgraphs and is necessary for presenting a compact mathematical model in Section 3. This is achieved by using pairs of *(X)OR-nodes* enclosing the alternative subgraphs and connecting them to the remaining parts of the graph. The OR-nodes get task time 0 and are represented by triangles. The subprocess is started with an *entry OR-node* (visualized as a triangle with the peak at the left) and finalized with a *terminal OR-node* (triangle

1 In the usual nomenclature the problem has to be named ASALBP-1 (minimize m given c). Since other problem versions can be deduced in a simple manner (cf. Scholl 1999, ch. 2), we restrict to this version and omit the "1" indicating the version to ease presentation.

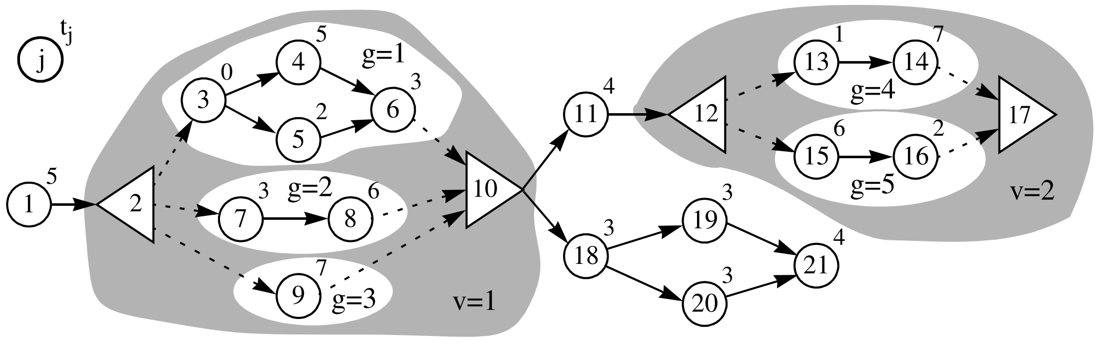

**Figure 1.** Example of a disjunctive precedence graph representing an ASALBP-instance

with the peak at the right). To keep the presentation lucid, it is moreover defined that each alternative subgraph must have a unique start and end node. If this is not already the case, dummy nodes with time 0 will be introduced. The (dummy) start nodes of all alternative subgraphs become successors of the entry OR-node, while the (dummy) end nodes become predecessors of the terminal OR-node. The respective arcs are called *disjunctive* (indicated by dashed lines) because they represent alternative branches. Furthermore, the predecessors of the subprocess are linked to the entry OR-node and the terminal OR-node is linked to the successors of the subprocess by normal arcs, respectively.

For a more formal description, we introduce the following notation:

| | |
|---|---|
| VS | set of variable subprocesses |
| $\langle e(v), t(v) \rangle$ | OR-node pair enclosing variable subprocess $v \in VS$ |
| SG(v) | set of alternative subgraphs for subprocess $v \in VS$ |
| SG | set of all subgraphs; $SG = \bigcup_{v \in VS} SG(v)$ |
| PN | number of different production processes; $PN = \prod_{v \in VS} \lvert SG(v) \rvert$ |
| V(g) | set of tasks contained in subgraph g |
| g=0 | fixed "subgraph" of all tasks outside variable subprocesses |
| g(j) / v(j) | subgraph/variable subprocess to which task j belongs ($v(j) := 0$ if $g(j) = 0$) |
| ts(J) | total task time of (sub)graph with node set J; $ts(J) = \sum_{j \in J} t_j$ |

Figure 1 shows an example with two variable subprocesses, i.e., $VS = \{1,2\}$. The first one ($v = 1$) is enclosed by the OR-node pair $\langle e(1), t(1) \rangle = \langle 2, 10 \rangle$ and has three alternative subgraphs with different task sets, task times and precedence relations, $SG(1) = \{1,2,3\}$. The subgraph $g = 1$ contains the four tasks in $V(1) = \{3,4,5,6\}$ one of which is a dummy start node. In the second subgraph the number of operations is reduced, e.g., by using an advanced tool ($V(2) = \{7,8\}$), while the third consists of a single task which might be the installation of a premounted module as a whole ($V(3) = \{9\}$). The variable subprocess 2 with OR-node pair $<12,17>$ has two alternatives, i.e., $SG(2) = \{4,5\}$ with $V(4) = \{13,14\}$ and $V(5) = \{15,16\}$, and represents, e.g., sequence-dependent task times (13 and 16 as well as 14 and 15 might represent the same work content but require different times due to mutual impediment, respectively). The remaining tasks are fixed, i.e., $V(0) = \{1,11,18,19,20,21\}$.

The problem, which is characterized as $[pa^{subgraph}| \, |m]$ in the classification scheme of Boysen et al. (2007), can now be stated as follows: For each variable subprocess exactly one of the alternative subgraphs is to be chosen, i.e., exactly one of the dashed arcs emerging from each entry node must be followed (the corresponding dashed arc entering the terminal node is reached automatically), while the other arcs and the enclosed subgraphs are to be ignored. All tasks *activated* in such a manner have to be assigned to a minimal number of stations such that the cycle time and the precedence constraints (relating activated tasks) are observed.

In our example, $PN = 6$ (complete) production processes can be obtained by combining the three alternatives of $v = 1$ and the two alternatives of $v = 2$. Given the cycle time $c = 10$, the combination of the subgraphs 1 and 4 (production process p=1) is optimal. The station loads $S_1 = \{1, 4\}$, $S_2 = \{5, 6, 11, 13\}$, $S_3 = \{14, 18\}$, and $S_4 = \{19, 20, 21\}$ constitute the optimal line balance with $m^* = 4$ stations (OR-nodes, dummy tasks and deactivated tasks are omitted).

An equivalent presentation of the problem is obtained by merging all variable subgraphs to form a single OR-node pair enclosing the whole graph. This is achieved by combining the different alternatives of all variable subprocesses as described before. Figure 2 shows this transformed graph for our example. In order to ease presentation, the terminal nodes (overall and per processing alternative) are omitted and the original node numbers are preserved. Generally, it is always possible to give different equivalent representations of the same problem instance. However, in case of many processing alternatives it is recommendable to use a node and space sparing precedence graph by using many OR-node pairs. It is even possible and sometimes useful to nest them, i.e., one variable subgraph is part of an alternative of an enclosing variable subgraph.

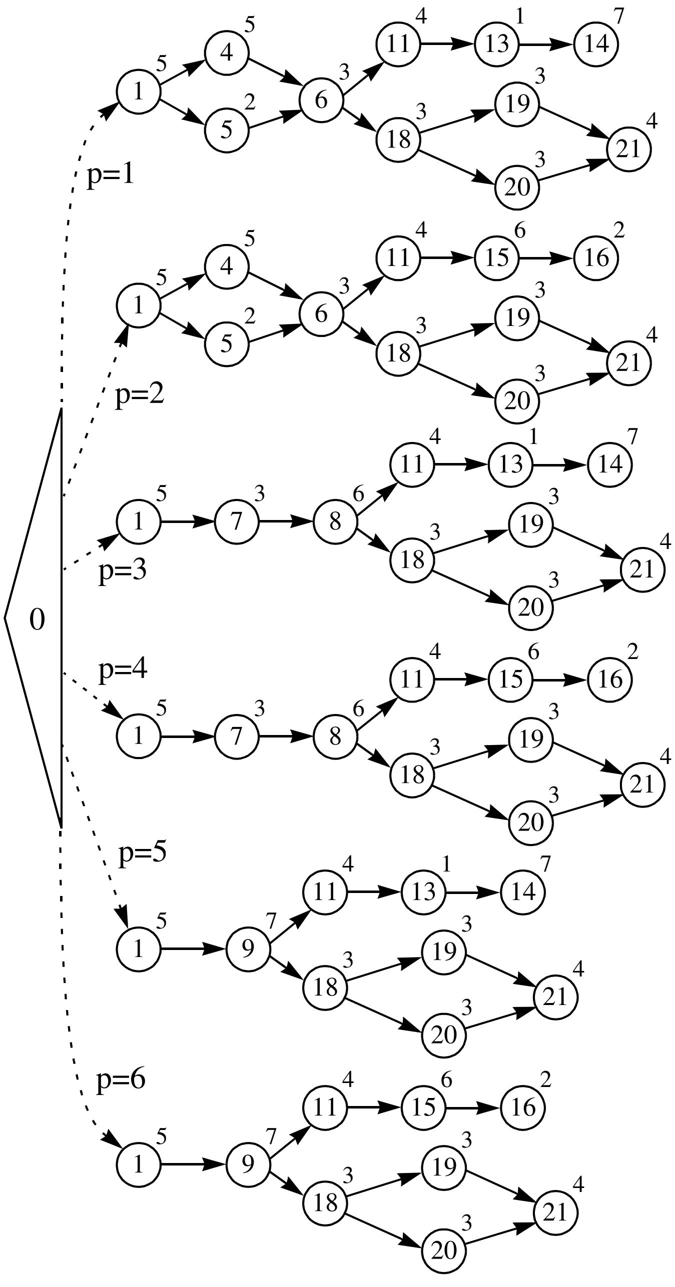

**Figure 2.** Graph with complete production processes

*Remarks:*

- If there exists a task j with $t_j > c$ in any alternative subgraph, this subgraph cannot be part of a feasible solution and is discarded. However, if a task time of any task in V(0) exceeds the cycle time or all alternative subgraphs of the same variable subprocess have been discarded, no feasible solution exists at all. In the following, we assume that no cycle time exceeding tasks exist (anymore).

- The example shows that it is not necessarily optimal to choose the producion process with minimal total task time as is usually recommended (cf. Capacho and Pastor 2006).

## 3 Mathematical model for ASALBP

Capacho and Pastor (2005, 2007) have formulated two binary programs for ASALBP. We present a modified model with significantly less binary variables. It is an extension of the standard formulation for SALBP (cf. Bowman 1960; White 1961; Thangavelu and Shetty 1971; Patterson and Albracht 1975; Scholl 1999, p. 29).

We assume that a node saving representation of the problem is given and that the graph has single start node 1 and terminal node n (both real or dummy tasks, if necessary). The set of nodes V = {1,...,n} consists of four disjunctive subsets for the different node types:

| | | | |
|---|---|---|---|
| $V_r$ | set of real tasks | $V_s$ | set of entry nodes |
| $V_t$ | set of terminal nodes | $V_d$ | set of dummy tasks with duration 0 |

The model only requires *binary assignment variables* $x_{jk}$ defined as follows ($\overline{m}$ is an upper bound on the number of stations; $\overline{m} \le n$):

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

In order to reduce the number of variables, earliest and latest stations are computed. However, the traditional approach for SALBP (cf. Saltzman and Baybars 1987) must be modified due to disjunctive parts of the precedence graph. Thus, a previous step is necessary to compute for each task j a lower bound on the total task time of all predecessors and successors, respectively. This is achieved as follows:

(1) For each variable subprocess $v \in VS$, a lower bound on the total task time is computed as $\underline{ts}(v) = \min\{ts(V(g)) \mid g \in SG(v)\}$.

(2) For each task $j \in V$ temporarily modify the precedence graph such that all subprocesses $v \in VS - v(j)$ are substituted by a single task with task time $\underline{ts}(v)$ which inherits the precedence relations to predecessors from e(v) and the relations to successors from t(v). From this modifed graph retrieve the set $P^*(j)$ of all predecessors and $F^*(j)$ of all successors. Based on *relative task times* $\tau_j = t_j / c$, compute earliest stations $E_j$ and latest station $L_j$:

$$E_j = \left\lceil \tau_j + \sum_{h \in P^*(j)} \tau_h \right\rceil, \; L_j = \overline{m} + 1 - \left\lceil \tau_j + \sum_{i \in F^*(j)} \tau_i \right\rceil \tag{2}$$

Task j can only be assigned to one of the stations in the *station interval* $SI_j = [E_j, L_j]$. So, only a subset $B_k = \{j \in V \mid k \in SI_j\}$ of the tasks are *potentially assignable* to the stations k = 1,...,$\overline{m}$. Using the station intervals the number of the variables can be reduced, because $x_{jk}$ has only to be defined for $j \in V$ and $k \in SI_j$.

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

$$z_j := \sum_{k \in SI_j} k \cdot x_{jk} \qquad \text{for } j \in V \tag{3}$$

The binary program for ASALBP is given by objective (4) and constraints (5)-(15). The assumption that n is a single sink node of the precedence graph allows for a simple determination of the number of stations actually required such that minimizing $z_n$ is sufficient instead of the more complex term and the additional binary station indicators used by Capacho and Pastor (2005, 2006).

$$\text{Minimize } m(\mathbf{x},\mathbf{z}) = z_n \tag{4}$$

subject to

$$\sum_{k \in SI_1} x_{1k} = 1 \tag{5}$$

$$\sum_{k \in SI_j} x_{jk} \le 1 \qquad \text{for all } j \in V-\{1\} \tag{6}$$

$$\sum_{j \in B_k} t_j \cdot x_{jk} \le c \qquad \text{for } k = 1,...,\overline{m} \tag{7}$$

$$z_j = \sum_{k \in SI_j} k \cdot x_{jk} \qquad \text{for all } j \in V \tag{8}$$

$$z_i - z_j \le 0 \qquad \text{for all } i \in V-V_s,\ j \in F_i \mid SI_i \cap SI_j \ne [\,] \tag{9}$$

$$z_i - z_j \le \overline{m} \cdot \left(1 - \sum_{k \in SI_j} x_{jk}\right) \qquad \text{for all } i \in V_s,\ j \in F_i \mid SI_i \cap SI_j \ne [\,] \tag{10}$$

$$\sum_{k \in SI_i} x_{ik} - \sum_{k \in SI_j} x_{jk} \ge 0 \qquad \text{for all } j \in V-V_t \text{ and } i \in P_j \tag{11}$$

$$\sum_{j \in F_i} \sum_{k \in SI_j} x_{jk} - \sum_{k \in SI_i} x_{ik} = 0 \qquad \text{for all } i \in V_s \tag{12}$$

$$\sum_{i \in P_j} \sum_{k \in SI_i} x_{ik} - \sum_{k \in SI_j} x_{jk} = 0 \qquad \text{for all } j \in V_t \tag{13}$$

$$x_{jk} \in \{0,1\} \qquad \text{for } j \in V \text{ and } k \in SI_j \tag{14}$$

$$z_j \ge 0 \qquad \text{for } j \in V \tag{15}$$

As an initialization, constraints (5) assure that the overall start node (real, dummy or even entry OR-node) gets assigned to exactly one station. For the other tasks it is, due to (11)-(13), sufficient to guarantee that they are assigned to at most one station as formulated in (6). The cycle time constraints are given in (7), the auxiliary variables are defined in (8). The precedence constraints (except for the disjunctive arcs emerging from entry OR-nodes) are represented by (9). Concerning a disjunctive arc $i \in V_s$, the precedence constraint $(i,j)$ must be fulfilled only if the subgraph $g(j)$ is selected. This is achieved by (10), which is activated only if task j is assigned to a station.

By (11) it is ensured that a task j (except for terminal OR-nodes) with an unassigned predecessor cannot be assigned to a station, because it is not in an activated part of the graph in such a case. Equations (12) state that exactly one of the immediate successors (one of the disjunctive arcs) is chosen for each reached entry OR-node and that none is chosen if the entry OR-node is not in an active part of the graph. Similarly, (13) enforce that the terminal OR-node is only assigned if the corresponding subprocess is activated, i.e., one of the nodes terminating the contained subgraphs is assigned. By propagating the activation status of nodes through the graph in the manner described, the model gets by without binary variables indicating the activation status of subgraphs as used by Capacho and Pastor (2006).

Compared to the model CP of Capacho and Pastor (2006), the numbers of binary variables are reduced significantly, because CP includes (up to)[2] $\overline{m} \cdot n \cdot |SG|$ assignment variables, $\overline{m} - LB$ station indicators (LB is a lower bound on the number of stations) and $|SG|$ subgraph selection variables while the new model gets by with (up to) $\overline{m} \cdot (n+|SG|)$ assignment variables. Similarly, CP needs more restrictions. Using the O-notation, the order of magnitudes are as given in Table 1. Utilizing the relations $\overline{m} \le n$ and $|SG| \le n$, CP has $O(n^3)$ variables and constraints, while the new model only requires $O(n^2)$ variables and constraints.

| | # variables | # constraints |
|---|---|---|
| CP | $O(\overline{m} \cdot n \cdot \lvert SG \rvert)$ | $O(n^2 \cdot \lvert SG \rvert + \overline{m})$ |
| new | $O(\overline{m} \cdot (n+\lvert SG \rvert))$ | $O((n+\lvert SG \rvert)^2+\overline{m})$ |

**Table 1.** Comparison of model sizes

## 4 A SALBP-based search procedure for ASALBP

Though the improved model presented in Section 3 should considerably reduce the solution times of standard MIP solvers, it is also not able to solve problem instances of real-world size to optimality in reasonable computation times. This has been detected by preliminary tests confirming the fundamental results of Capacho and Pastor (2006). Only small instances with up to about 50 instances are solvable somewhat efficiently.

As a consequence, it is necessary to construct adequated solution procedures. In the following, we present and discuss an approach to solve ASALBP without developing highly specialized procedures. Instead, we show how to use existing solution procedures for SALBP in an effective manner. The resulting solution method can be used as an exactly optimizing and as an heuristic procedure as well, thus offering valuable support in all decision situations.

The solution approach extends the one proposed for the ALBP with sequence-dependent task times (SDALBP) of Scholl et al. (2006) and is based on the relationships between ASALBP and SALBP. In order to keep the paper short, we only shortly describe the theoretical basics and features of the procedure which can be taken over from this former approach and concentrate on aspects to be modified.

Recalling the transformed precedence graph in Figure 2, it becomes obvious that ASALBP consists of PN disjunctive production processes (=*included* SALBP instances) one of which is to

2 Note that in both cases the number of assignment variables is reduced by means of station intervals. Moreover note that the auxiliary variables $z_j$ can be simply eliminated.

be selected (cf. Capacho and Pastor 2006). As a consequence, a simple solution approach for ASALBP is to build and solve all included SALBP instances and take (one of) the SALBP solution(s) with minimal number of stations. Of course, this approach can be rather inefficient, because SALBP is an NP-hard problem itself and PN might be very large as it requires completely combining all possible subgraphs. Facing this fact, Capacho and Pastor (2005) take it for granted that solving ASALBP by iteratively solving the underlying SALBP instances is not useful at all. This assumption might be justified for the just mentioned uninformed solution approach. However, by using an intelligent search method, the SALBP-based approach can be enhanced considerably. Furthermore, it has to be considered that the derived SALBP instances are much smaller than the entire graph, because all deactivated subgraphs are eliminated.

As solution method for solving the included SALBP instances, we apply the branch-and-bound procedure SALOME of Scholl and Klein (1997) which is supposably the best exact solution method available for SALBP (cf. Sprecher 1999; Scholl and Klein 1999, Scholl 1999).

## 4.1 Bound computation

A *lower bound* on the number of stations can be computed by constructing a SALBP instance as described in Section 3 in the context of computing station intervals: Each variable subprocess $v \in VS$ is substituted by a single task with task time $\underline{ts}(v)$ such that the modified graph represents the overall minimum time requirement. Applying the well-known capacity bound for SALBP (cf. Baybars 1986), we get the lower bound:

$$LB1 = \left\lceil ts(0) + \sum_{v \in VS} \underline{ts}(v) \right\rceil \tag{16}$$

Additionally, we may compute lower bound values $LB(p)$ for all included SALBP instances p (using the complete arsenal of bound arguments for SALBP in each case; cf. Scholl and Becker 2006). The minimum of these values serves as a second lower bound for ASALBP:

$$LB2 = \min\{LB(p) \mid p = 1,...,PN\} \tag{17}$$

Since LB1 refers to one of the included instances, where the time minimizing subgraph is chosen in each variable subprocess, the relation $LB2 \geq LB1$ holds. However, computing LB2 is much more expensive such that LB1 may serve as a reasonable start value.

The best (maximal) lower bound value known in a certain process step is called *global lower bound* LB.

*Upper bounds* on the number of stations are given by feasible solutions to any included SALBP instance. As *global upper bound* UB we denominate the best known (minimal) upper bound. As initial value, we might use $UB1 = |V(0)| + \sum_{v \in VS} \min\{|V(g) \cap V_r| \mid g \in SG(v)\}$, because a trivial solution is obtained by choosing the subgraph with least number of tasks for each variable subprocess and assigning each task to an own station. The solution that corresponds to UB is called *incumbent solution*.

*Bounding rules:* The bounds can be used to accelerate the search process in several manners:

- BR1: Each included SALBP instance p with $LB(p) \geq UB$ is discarded, because its optimal solution cannot provide an overall improvement.

- BR2: Whenever a feasible solution with objective value $UB(p)<UB$ is obtained for any SALBP instance p, the global upper bound is improved by $UB:=UB(p)$.
- BR3: The complete procedure is immediately terminated whenever $LB=UB$ is obtained, because the incumbent solution is optimal, i.e., $m^*=UB$.
- BR4: When SALOME is applied to an instance p, it almost immediately finds a first feasible solution and, thus, a first upper bound UB(p) due to the depth-first search performed. Furthermore, it successively strengthens the lower bound LB(p) due to the local lower bound method contained. In many cases, SALOME can terminate before finding the optimal objective value $m^*(p)$ of instance p. This is possible whenever $LB(p) \geq UB$ holds.

## 4.2 Search procedure

Using the bounding options and the power of SALOME as discussed before, an versatile search procedure has been constructed by adapting the (advanced) search procedure of Scholl et al. (2006). In order to keep the presentation short, we give a tightened description without specifying each termination condition in detail. As a rule, the entire search procedure or the application of SALOME to an instance p is always stopped whenever one of the bounding rules described above applies. The following steps have to be performed; for details see Scholl et al. (2006):

(1) Initialize the search by computing the initial global bounds LB := LB1 and UB :=UB1.

(2) Generate the solvable instances p = 1,...,PN in a systematic (lexicographic) manner. For each just generated instance p, apply SALOME for a very short time limit TL1 but at least until it has found a first feasible solution. This provides a lower bound LB(p) and an upper bound UB(p). If instance p is not discarded by applying the bounding rules BR1 to BR4, it is stored in a list L, together with the current bound values LB(p) andUB(p).

(3) Adjust the list L through BR1. If the remaining L is empty, then terminate the complete procedure. Otherwise, sort the list according to non-decreasing lower bound values LB(p) with ties broken in favour of instances with smaller total task times (second level tie breaker: original instance numbers).

(4) Work through the list instance-per-instance, let p be the current instance. If $LB<LB(p)$, then increase LB to LB(p), because p has the smallest lower bound value of all remaining instances due to the sorting of the list. If BR3 does not hold, apply SALOME to p starting with the stored LB(p) and UB(p). Apply BR2 and BR3 and remove p from L.

*Result:* Optimal solution to ASALBP with m* := UB and the task assignments stored as incumbent solution. If the procedure has stopped due to a time limit, it has only found a feasible (incumbent) solution with UB stations and a lower bound LB.

## 5 Computational experiments

In order to examine the performance of the search procedure described in Section 4, we perform computational experiments based on systematically constructed benchmark test instances.

## 5.1 Test data generation

As a basis for generating test instances for ASALBP we use the well-known data set for SALBP with 269 instances based on 25 precedence graphs having 8 to 297 nodes each connected to several cycle times. For a detailed description of the SALBP data set and the characteristics of the underlying precedence graphs see Scholl (1999, ch. 7.1) and the "homepage for assembly line optimization research" *www.assembly-line-balancing.de*, where the corresponding data files can be downloaded from.

To generate different classes of instances, we use the parameter pr which defines the portion of variable subprocesses on the number of original tasks (in percent). Due to the combinatorial explosion, not more than 20 subprocesses are allowed such that $|VS| = \min\{20, (pr \cdot n) / 100\}$.[3] Four data sets are generated by setting pr to 2, 5, 7, and 10. This amounts to 1,076 instances.

To define a variable subprocess v, a task j with $t_j > 0.2 \cdot c$ $(\tau_j > 0.2)$ is chosen randomly. This task is thought to represent a high-level task integrating several elementary operations. It is used as the first subgraph of v. Alternative subgraphs are built by using the building blocks depicted in Figure 3. The grey nodes without indexes are dummy nodes with zero task time.

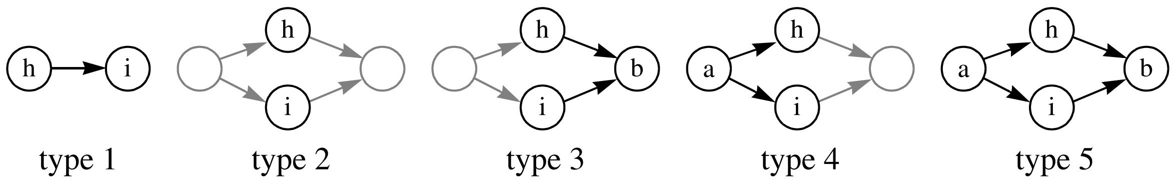

**Figure 3.** Building blocks for generating alternative subgraphs

A further assumption concerns the task times in the alternative subgraphs. It is based on the observation that a single task will be never chosen if its task time is not smaller than the total task time of an alternative subgraph with several tasks, because assigning smaller tasks to stations is easier than assigning large ones. Thus, it is assumed that a subgraph replacing a single task must have a larger total time. This assumption is rather realistic, because operations are integrated into a single task just for the reason of time reduction (e.g., installing a module instead of building the module while mounting its parts). Additionally, it must be controlled that the total time of a subgraph is not much larger than the single task to avoid the reverse domination problem. Furthermore, it seems to be reasonable that the types of building blocks chosen as alternative subgraph should depend on the value of $t_j$.

These reflections lead to a generation procedure which repeats the following steps for $v = 1,\ldots,|VS|$:

(1) Randomly select a task $j \in V_r$ with $\tau_j > 0.2$ which has not been chosen before.[4] In the precedence graph, replace j by an OR-node pair $\langle e(v), t(v) \rangle$ with zero task times. As the first alternative in SG(v), task j is set between the OR-nodes. Determine the number of alternatives a(v) (including j) by randomly drawing from {2,3,4}.

3 The restriction to 20 subprocesses only applies for the largest precedence graph with 297 tasks and $pr \geq 0.7$.

4 This and all other random choices are based on uniformly distributed standard random numbers. Subgraph types are randomly chosen with repetition.

(2) Construct $a(v)-1$ alternative subgraphs each of which is placed in parallel to task j. The available types and the restrictions on task times depend on the value of $\tau_j$. The individual task times are chosen such that they belong to the interval $\left[\frac{1}{5}\cdot t_j, \frac{4}{5}\cdot t_j\right]$. Each added node gets the number $n:=n+1$.

   (a) If $\tau_j \le 0.35$, then for each subgraph randomly select type 1 or 2 (two tasks h and i; Figure 3) and randomly generate task times $t_h$ and $t_i$ such that $t_j \le t_i + t_h \le \frac{5}{4}\cdot t_j$ holds.

   (b) If $0.35 < \tau_j \le 0.5$, then for each subgraph randomly select type 1 to 4 (two or three tasks) and randomly generate task times such that their sum falls into the interval $\left[t_j, \frac{4}{3}\cdot t_j\right]$.

   (c) If $\tau_j > 0.5$, then for each subgraph randomly select type 1 to 5 (two to four tasks) and randomly generate task times such that their sum falls into the interval $\left[t_j, \frac{3}{2}\cdot t_j\right]$.

(3) After having generated all subprocesses, the graph is re-numbered in a topological order.

In step (1) it is possible to select a task which has been added in an iteration before such that nested variable subprocesses are generated and, thus, larger subgraphs with more than four tasks can occur.

The resulting four data sets can be downloaded from *www.assembly-line-balancing.de*.

## 5.2 Comparing the advanced search procedure with other approaches

We compare the developed advanced search procedure (AS) with two other approaches:

- Sequential planning (SP): Select the SALBP instance with minimal total task time in the first step and solve this instance in the second step (applying SALOME with the same time limit as is given to the other procedures). This represents the usual managerial approach when the production process is determined before balancing.
- Basic search method (BS): The SALBP instances are generated and solved one after the other using SALOME. To restrict the search, only the simple lower bound LB1 can be used as a global lower bound. No individual time limit is given to SALOME per SALBP instance, i.e., each instance is solved to optimality (starting with $UB(p) = UB$). The complete process is stopped when the time limit is reached.

All approaches were coded in Borland Delphi 7.0. The experiments were run on a personal computer with an AMD Athlon processor of 1.4 GHz clock speed and 768 MByte of RAM. For each instance a time limit of 1,800 seconds was imposed.

Table 2 summarizes the results based on the following measures:

| | |
|---|---|
| ∅n | average number of tasks (including OR-nodes and dummy nodes) |
| ∅nt | average number of real tasks |
| #prov | number of optima found and proven (out of 269 instances) |
| #opt | number of (known) optima found (out of 269 instances) |
| ∅dev | average relative deviation from minimal number of stations (or best known lower bound if the optimum is still unknown) in % |
| ∅gap | average relative gap between upper and (procedure specific) lower bound in % |
| ∅cpu | average computation time (contains a value of 1,800 seconds for time out cases) |

| | | | SP | | | | BS | | | | | AS | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| pr | ∅n | ∅nt | #opt | ∅dev | ∅gap | ∅cpu | #prov | #opt | ∅dev | ∅gap | ∅cpu | #prov | #opt | ∅dev | ∅gap | ∅cpu |
| 2 | 111 | 95 | 232 | 0.39 | 5.05 | 103.8 | 251 | 251 | 0.18 | 0.34 | 129.5 | 256 | 256 | 0.13 | 0.13 | 101.6 |
| 5 | 135 | 97 | 226 | 0.62 | 5.05 | 103.8 | 235 | 235 | 0.36 | 0.42 | 245.8 | 234 | 234 | 0.37 | 0.37 | 245.7 |
| 7 | 150 | 99 | 224 | 0.90 | 5.05 | 103.8 | 225 | 225 | 0.51 | 0.61 | 311.6 | 233 | 234 | 0.38 | 0.39 | 247.4 |
| 10 | 165 | 101 | 216 | 1.25 | 5.05 | 103.8 | 216 | 216 | 0.60 | 0.65 | 437.9 | 220 | 222 | 0.54 | 0.57 | 395.8 |
| total | 140 | 98 | 898 | 0.79 | 5.05 | 103.8 | 927 | 927 | 0.41 | 0.51 | 281.2 | 943 | 946 | 0.36 | 0.37 | 247.6 |

**Table 2.** Summary of the results

Due to the systematic construction of the test instances, the original SALBP instance is contained in each derived ASALBP instance and shows minimal total task time among all included instances. Thus, the sequential planning approach (SP) has to solve the same (the original) SALBP instance in all four data sets. Because it cannot obtain another global lower bound than LB1, the average gap between upper and lower bound is rather large such that no useful information on the solution quality can be derived from applying this procedure alone. Compared to the search procedures the solution quality is considerably lower. While all procedures optimally solve the small-sized and easy instances quickly, the search procedures solve much more difficult instances of real-world size to optimality. This result demonstrates the necessity to consider processing alternatives and balancing in a simultaneous decision. It is expected that the effect is even larger if real-world instances are considered, because the generated instances have a well-defined structure with rather small alternatives and moderate differences in task times.

A comparison of the search procedures reveals that already the basic procedure BS provides good results but often fails in closing the bound gap, because it cannot enlarge the global lower bound in a systematic manner like the improved search procedure AS. Moreover, it requires more computation time.

Concercing parameter pr, it becomes obvious that solving ASALBP instances with more variable subprocesses is considerably more complex though the number of real nodes is not enlarged considerably. However, the number of subgraphs to be selected and, thus, the number of SALBP instances to be solved grows exponentially with increasing pr. As a consequence, the differences between the procedures decline as only a small subset of SALBP instances can be considered in the time available. While BS finds almost the same number of optimal solutions as SP for pr=7 and pr=10, AS finds some additional optima. Considering the average deviation from optimum shows that both search procedures find near-optimal solutions also in time-out cases while SP frequently comes out with inferior solutions.

Finally, to support the supposition that solving ASALBP instances with standard MIP solvers is not recommendable, we performed some limited tests. For example, an instance with 48 task was solved by XPress MP in 135 seconds while it took only 0.07 seconds with AS. Similar results are obtained for other small- and medium-sized instances. For large-sized instances, even a feasible solution could not be obtained in a reasonable time span.

## 6 Conclusions and future research

In this paper, we have considered the alternative subgraphs assembly line balancing problem which simultaneously decides on used production processes and assigns task to stations to optimize line efficiency. It has been demonstrated that the simultaneous consideration is superior to the traditional sequential planning approach. Furthermore, the problem has been formalized and an improved mathematical model has been given. For solving the problem, a search procedure which is based on iteratively solving instances of the simple assembly line balancing problem has been developed.

A computational experiment indicates that even problem instances of real-world size can be solved to optimality or at least near optimal solutions are found by using the effective solution procedures available for SALBP, like SALOME, while standard MIP solvers are very inefficient.

The SALBP-based approach has a further interesting advantage, because it can be used for solving ASALBP versions with cost objectives in a straightforward manner by computing lower and upper bounds of included SALBP instances using the cost objective instead of the capacity based one. This extension is very relevant for practice, because processing alternatives will not only cause different operation times but also differ with respect to cost. As an example, mounting a complete module instead of combining its parts on the line reduces task times but will increase cost as it requires a pre-assembly outside the line or must be bought from a supplier.

We believe that a lot of further practice-relevant ALB extensions can be tackled by flexible search procedures based on the repeated solution of SALBP-instances. In the light of the high performance of existing solution procedures for SALBP, an identification of further ALB extensions with similar characteristics promises to be fruitful. It is very likely that the solution quality of such approaches will easily surpass standard MIP solvers and simple priority rule based procedures usually proposed for new problems, while the effort of implementation is reduced in comparison to more specialized solution procedures.

## Acknowledgement

For programming parts of the search procedure and the data generation procedure, we are indebted to Katrin Bley, a former student of Business Mathematics at Jena University.

## References

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

Becker, C., Scholl, A. (2006). A survey on problems and methods in generalized assembly line balancing. European Journal of Operational Research, 168, 694-715.

Bowman, E. (1960). Assembly-line balancing by linear programming. Operations Research, 8, 385-389.

Bukchin, J., Tzur, M. (2000). Design of flexible assembly line to minimize equipment cost. IIE Transactions, 32, 585-598.

Capacho, L., Pastor, R. (2005). ASALBP: The alternative subgraphs assembly line balancing problem. Working paper IOC-DT-P 2005-5, Universitat Politècnica de Catalunya.

Capacho, L., Pastor, R. (2006). The ASALB problem with processing alternatives involving different tasks: Definition, formalization and resolution. Lecture Notes in Computer Science 3982, 554-563.

Capacho, L., Pastor, R. (2007). ASALBP: The alternative subgraphs assembly line balancing problem. International Journal of Production Research (to appear).

Capacho, L., Pastor, R., Guschinskaya, O., Dolgui, A. (2006). Heuristic methods to solve the alternative subgraphs assembly line balancing problem. Automation Science and Engineering, 2006. CASE ´06. IEEE International Conference on, Shanghai, China, 501-506.

Capacho, L., Pastor, R., Dolgui, A., Gunshinskaya, O. (2007). An evaluation of constructive heuristic methods for solving the alternative subgraphs assembly line balancing problem. Journal of Heuristics (to appear).

Erel, E., Sarin, S.C. (1998). A survey of the assembly line balancing procedures. Production Planning & Control, 9, 414-434.

Patterson, J.H., Albracht, J.J. (1975). Assembly-line balancing: zero-one programming with fibonacci search. Operations Research, 23, 166-172.

Pinto, P.A., Dannenbring, D.G., Khumawala, B.M. (1983). Assembly line balancing with processing alternatives: An application. Management Science, 29, 817-830.

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

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.

Scholl, A. (1999). Balancing and sequencing of assembly lines. 2nd ed., Heidelberg: Physica.

Scholl, A., Becker, C. (2006). State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. European Journal of Operational Research, 168, 666-693.

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

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.

White, W.W. (1961). Comments on a paper by Bowman. Operations Research, 9, 274-276.