Sprecher, Arno

A competitive exact algorithm for assembly line balancing

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 449

Provided in Cooperation with:
Christian-Albrechts-University of Kiel, Institute of Business Administration

Suggested Citation: Sprecher, Arno (1997) : A competitive exact algorithm for assembly line balancing, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 449, Universität Kiel, Institut für Betriebswirtschaftslehre, Kiel

This Version is available at:
https://hdl.handle.net/10419/177310

Manuskripte

aus den

Instituten für Betriebswirtschaftslehre

der Universität Kiel

No. 449

A Competitive Exact Algorithm for Assembly Line Balancing [1]

Arno Sprecher

July 1997

Arno Sprecher, Institut für Betriebswirtschaftslehre, Christian-Albrechts-Universität zu Kiel, Olshausenstraße 40, 24098 Kiel, Germany.

Email: Sprecher@bwl.uni-kiel.de

WWW : http://www.wiso.uni-kiel.de/bwlinstitute/prod

FTP : ftp://www.wiso.uni-kiel.de/pub/operations-research

[1]Supported by the Deutsche Forschungsgemeinschaft

Abstract

In this paper we present a branch-and-bound algorithm for solving the simple assembly line balancing problem of type 1 (SALB-1). The algorithm relies on the precedence tree guided enumeration scheme introduced for solving a broad class of resource-constrained project scheduling problems. The general enumeration scheme ranks among the most powerful algorithms for solving the well-known single- and multi-mode resource-constrained project scheduling problem.

By reformulating the SALB-1 as a resource-constrained project scheduling problem with a single renewable resource the availability of which varies with time, the problem can be solved with the general algorithm. Only minor adaptations are necessary to implement an efficient assembly line balancing procedure. Taking into account SALB-1 specific information, the transparency of the enumeration scheme allows to generalize classical dominance concepts from Jackson and Johnson substantially.

The procedure has been coded in C and implemented on a personal computer. The computational results indicate that the algorithm can compete with the best algorithms currently available for solving the SALB-1.

Keywords: Assembly Line, Project Scheduling, Resource Constraints, Model, Branch-and-Bound, Heuristic, Computational Results.

1 Introduction

The assembly line balancing problem considers the assignment of tasks of different durations to stations (cf. Baybars [1], Hoffmann [5], Jackson [6], Johnson [7], Scholl and Klein [14]). Precedence relations between some of the tasks impose a partial ordering, reflecting which task has to be completed before others. The precedence relations can be depicted by an acyclic network with exactly one source and sink. The tasks are related to the assembly of a product to be performed at consecutive stations. A conveyor belt moves the product from station to station. At the stations the assigned tasks have to processed within the cycle time, i.e., within the time the conveyor stops at the station, or, as in automotive industry within the time the product moves within the station. Given the cycle time the production rate, i.e., the number of products to be finished per period, is fixed, and vice versa. In the simple assembly line balancing problem (SALB) the processing times of the tasks are known in advance and cannot be split between different stations.

Two types of the SALB are mainly considered: First, the SALB of type-1 (SALB-1) is involved with the minimization of the number of stations for a given cycle time. If, e.g, the stations are manned by a worker, the objective can be interpreted as the minimization of labor costs, too. Second, in the SALB of type-2 (SALB-2) the number of stations is fixed and the production rate is to be maximized,

or equivalently, the cycle time is to be minimized (cf. [8]).

We are concerned with the SALB-1. The problem is dealt with an adaptation of a simple and general algorithm developed for the resource-constrained project scheduling problem (RCPSP) with multiple modes. We describe how to model the SALB-1 as an RCPSP with a single mode and a single resource of time varying availability. Moreover, we discuss differences and similarities of the SALB-1 and the RCPSP. The basic branch-and-bound algorithm for the RCPSP is adapted to solve the SALB-1 and enhanced by search reduction schemes. The transparency of the enumeration process allows us to extend classical SALB-1 specific dominance rules from Jackson (cf. [6]) and Johnson (cf. [7]). The effect is substantial. Moreover, some bounds proposed in the literature have been strengthened and reduced in their complexity.

We proceed as follows: In Section 2 we briefly recapitulate the basic terminology of assembly line balancing. Moreover, the formal definition of the SALB-1 is presented. In Section 3 we summarize the RCPSP, and then model the SALB-1 in Section 4 as a single-mode resource-constrained project scheduling problem with a single renewable resource the availability of which varies with time. In Section 5 we present the branch-and-bound algorithm. The basic enumeration scheme is presented in Subsection 5.1, and the concepts for search tree reduction are discussed in Subsection 5.2. Differences and similarities between the SALB-1 and RCPSP are studied. In Section 6, we, first, recapitulate the bounds developed for the SALB-1, and, second, give some hints to strengthen their effect and reduce their computational complexity. In Section 7 we reveal the results of our computational analysis. Finally, in Section 8 we draw our conclusions for future research.

2 The SALB-1

We consider the assembly (production) of a product that can be decomposed into J tasks. The tasks have deterministic processing times t_j, $j = 1, \ldots, J$, and are, due to technological requirements, partially ordered by precedence relations. The precedence relations are described by the sets of predecessors $\mathcal{P}_j$ of the tasks j, $j = 1, \ldots, J$, indicating that a task j may not be processed before all the tasks h, $h \in \mathcal{P}_j$, are completed. The related graph can be depicted by an acyclic task-on-node network where the nodes represent the tasks and the arcs the precedence relations. We assume that the tasks are numerically labeled, i.e., a task always has a higher number than all its predecessors. W.l.o.g. the network has exactly one source $j = 1$ and one sink $j = J$. The start and finish task have, w.l.o.g, a zero duration. The tasks have to be assigned to linearly ordered stations.

For a given upper bound $\overline{V}$ on the number of stations, we denote the set of tasks assigned to station v,

i.e., the load of station v by LS_v, $v = 1, \ldots, \overline{V}$. If two tasks i and j with $i \in \mathcal{P}_j$ are assigned to stations v and $\overline{v}$, i.e., $i \in LS_v$ and $j \in LS_{\overline{v}}$ then $v \leq \overline{v}$ is required. The processing time of a station v is determined by the sum of processing times of the tasks assigned to the station v, it is denoted as $PT(LS_v)$. The processing time of the stations is limited by the cycle time c. Given the production rate, we can calculate the cycle time c, and vice versa. The objective is the minimization of the production costs given by the number of stations. The parameters are summarized in Table 1.

J	:	number of tasks
t_j	:	processing time of task j
$\overline{V}$	:	upper bound on the number of stations
$\mathcal{P}_j$ ($\mathcal{S}_j$)	:	set of immediate predecessors (successors) of task j
$\overline{\mathcal{P}}_j$ ($\overline{\mathcal{S}}_j$)	:	set of predecessors (successors) of task j within the transitive closure of the network
c	:	cycle time

Table 1: Problem Parameters – SALB-1

Obviously, the problem is infeasible, if there is a task having a processing time which exceeds the cycle time c. Otherwise, an upper bound $\overline{V}$ on the minimal number of stations is given by the number of tasks to be performed, i.e., $\overline{V} = J$. Since feasibility can be easily verified, we can use it and introduce binary variables x_{jv}, $j = 1, \ldots, J$, $v = 1, \ldots, \overline{V}$,

$$x_{jv} = \begin{cases} 1 \quad , & \text{if task } j \text{ is assigned to station } v \\ 0 \quad , & \text{otherwise.} \end{cases}$$

and obtain the mathematical programming formulation displayed in Table 2 (cf. [3], [4], [22], [10]). Since we have only one finishing task the objective (1) realizes the minimization of the number of stations. (2) ensures that each task is assigned to exactly one station. (3) guarantees, that the sum of the processing times of the tasks assigned to one station does not exceed the cycle time c. The precedence relations are taken into account by (4).

Note, we have assumed that the tasks assigned to one station have to be performed consecutively. Therefore, the precedence relations between a task h and a task j, $h \in \mathcal{P}_j$, that are not taken into account by assigning task h to a lower numbered station than task j, have to be considered by first processing task h and then task j at the station the tasks are assigned to. Due to numerically labeling of the network, the tasks assigned to one station can be precedence feasibly arranged by ordering them with respect to increasing labels.

$$\text{Minimize } Z(x) = \sum_{v=1}^{\overline{V}} v \cdot x_{Jv} \tag{1}$$

s.t.

$$\sum_{v=1}^{\overline{V}} x_{jv} = 1 \qquad j = 1, \ldots, J \tag{2}$$

$$\sum_{j=1}^{J} t_j \, x_{jv} \leq c \qquad v = 1, \ldots, \overline{V} \tag{3}$$

$$\sum_{v=1}^{\overline{V}} v \cdot x_{hv} \leq \sum_{v=1}^{\overline{V}} v \cdot x_{jv} \qquad j = 2, \ldots, J, h \in \mathcal{P}_j \tag{4}$$

$$x_{jv} \in \{0,1\} \qquad j = 1, \ldots, J, \; v = 1, \ldots, \overline{V} \tag{5}$$

Table 2: The Simple Assembly Line Balancing Problem of Type 1 – SALB-1

3 The RCPSP

In this section we briefly summarize the RCPSP with time-varying availability. The model will be used in the following section to formulate the SALB-1 as an RCPSP.

We consider a project which consists of J activities (jobs, tasks). Due to technological requirements precedence relations between some of the activities enforce that an activity j, $j = 2, \ldots, J$, may not be started before all its predecessors h, $h \in \mathcal{P}_j$, are finished. The structure of the project is depicted by a so-called activity-on-node (AON) network where the nodes and the arcs represent the activities and precedence relations, respectively. The network is acyclic and numerically labeled, that is an activity j has always a higher number than all its predecessors. W.l.o.g. activity 1 is the only start activity (source) and activity J is the only finish activity (sink). Both have a zero duration and do not request any resource.

Performing activity j, $j = 1, \ldots, J$, takes d_j periods and may not be preempted. Each period the activity is processed it requires certain amounts of the renewable resources. The set of renewable resources is denoted by R. Considering a horizon, that is, an upper bound $\overline{T}$ on the project's makespan, we have an available amount of K_{rt} units of renewable resource r, $r \in R$, in period t, $t = 1, \ldots, \overline{T}$. An activity j uses k_{jr} units of renewable resource r, $r \in R$, each period the activity is in process. The parameters are summarized in Table 3 and assumed as integer-valued. The objective is to find a makespan minimal schedule that meets the constraints imposed by the precedence relations and the limited resource availabilities.

J	:	number of activities
d_j	:	duration of activity j
R	:	set of renewable resources
$\overline{T}$	:	upper bound on the project's makespan
$K_{rt} \geq 0$	:	number of units of renewable resource r, $r \in R$, available in period t, $t = 1, \ldots, \overline{T}$
$k_{jr} \geq 0$	:	number of units of renewable resource r, $r \in R$, used by activity j each period it is in process
$\mathcal{P}_j$ $(\mathcal{S}_j)$	:	set of immediate predecessors (successors) of activity j
ES_j (EF_j)	:	earliest start time (finish time) of activity j, calculated neglecting resource usage
LS_j (LF_j)	:	latest start time (finish time) of activity j, calculated by neglecting resource usage and taking into account the upper bound $\overline{T}$ on the project's duration

Table 3: Problem Parameters – RCPSP

Given an upper bound $\overline{T}$ on the project's makespan we can use the precedence relations to derive time windows, i.e. intervals $[EF_j, LF_j]$, with earliest finish time EF_j and latest finish time LF_j, containing the precedence feasible completion times of activity j, $j = 1, \ldots, J$, by traditional forward and backward recursion as performed by the metra potential method (MPM). Analogously, the interval $[ES_j, LS_j]$ bounded from below and above by the earliest start time ES_j and the latest start time LS_j, respectively, can be calculated to reflect the possible precedence feasible start times.

Using the time windows derived we can now state the problem as a linear program. It was similarly presented by Talbot (cf. [20]). We use binary decision variables x_{jt}, $j = 1, \ldots, J$, $t = EF_j, \ldots, LF_j$,

$$x_{jt} = \begin{cases} 1 & , \text{ if activity } j \text{ is completed at the end of period } t \\ 0 & , \text{ otherwise.} \end{cases}$$

The model is presented in Table 4 and referred to as the RCPSP. Since there is exactly one finish activity, the objective function (6) realizes the minimization of the project's makespan. Constraints (7) ensure that exactly one completion time is assigned to each activity. The precedence relations are taken into account by (8). (9) guarantees, that the per-period availabilities of the renewable resources are not exceeded.

The RCPSP can be easily modified to include alternative ways of activity execution. The duration of the so-called modes is a discrete function of the quantities of the resource used (cf. [20]).

We can now reformulate the SALB-1 as a single-mode resource-constrained project scheduling problem, where the cycle time is reflected by a single renewable resource.

$$\text{Minimize } \Phi(x) = \sum_{t=EF_J}^{LF_J} t \cdot x_{Jt} \tag{6}$$

s.t.

$$\sum_{t=EF_j}^{LF_j} x_{jt} = 1 \qquad j = 1, \ldots, J \tag{7}$$

$$\sum_{t=EF_h}^{LF_h} t \cdot x_{ht} \leq \sum_{t=EF_j}^{LF_j} (t - d_j) x_{jt} \qquad j = 2, \ldots, J, h \in \mathcal{P}_j \tag{8}$$

$$\sum_{j=1}^{J} k_{jr} \sum_{q=\max\{t,EF_j\}}^{\min\{t+d_j-1,LF_j\}} x_{jq} \leq K_{rt} \qquad r \in R, t = 1, \ldots, \overline{T} \tag{9}$$

$$x_{jt} \in \{0,1\} \qquad j = 1, \ldots, J, t = EF_j, \ldots, LF_j \tag{10}$$

Table 4: The Resource-Constrained Project Scheduling Problem – RCPSP

4 The RCPSP Formulation of the SALB-1

Each task j, $j = 1, \ldots, J$, is considered as an activity, the durations of which are given by the corresponding processing times. The precedence relations between the activities are the ones, induced by the one-to-one correspondence of the tasks and the activities. We define $\overline{T} := \overline{V} \cdot (c + 1)$ and introduce a single renewable resource, i.e.,$R = \{1\}$ with an availability of

$$K_{1t} = \begin{cases} 0 & , \text{ if } t = v(c+1), v = 1, \ldots, \overline{V} \\ 1 & , \text{ otherwise} \end{cases}$$

Each activity j, $j = 1, \ldots, J$, uses one unit of resource 1 each period it is in process, i.e., $k_{j1} = 1$. The resource availability for $c = 5$ and $\overline{V} = 4$ is displayed in Figure 1. The objective is the minimization of the makespan of the newly derived project scheduling problem.

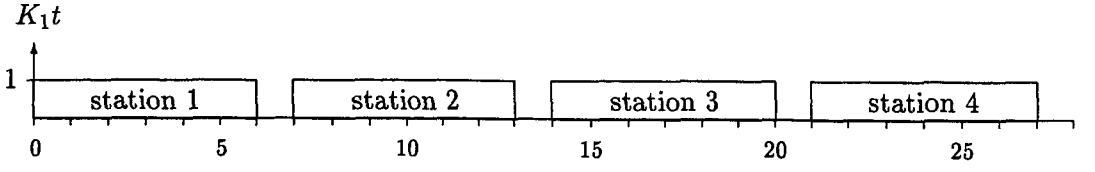

Figure 1: Resource Availability in the RCPSP Formulation of the SALB-1 – $c = 6$, $\overline{V} = 4$

The essential attribute is the time-varying supply. Since no unit of the renewable resource 1 is available in the periods t, $t = v(c+1)$, $v = 1, \ldots, \overline{V}$, and, moreover, each activity j, $j = 1, \ldots, J$, uses one unit

of the resource 1, the activities have to be performed in intervals $[(v-1)\cdot(c+1)+1, v\cdot(c+1)-1]$, $v=1,\ldots,\overline{V}$. From an optimal solution of the related project scheduling problem one can easily derive an optimal solution of the assembly line balancing problem. The activities are assigned to the stations related to the intervals the activities are completed (performed) in.

5 A Branch-and-Bound Algorithm for the SALB-1

As the RCPSP formulation of the SALB-1 in Section 4 suggests the problem can be considered as an RCPSP with a single resource with time-varying availabilities and activities using one unit each period they are in process. That is, only the resource availabilities and resource usages have to be generated appropriately and the problem can be solved by any RCPSP-algorithm capable of dealing with the time varying availabilities (cf. [16], [17], [18]). However, clearly, the foregoing is not reasonable, since unnecessary tests of obviously infeasible start times and unnecessary updates of the resource availabilities of the general scheme would reduce the efficiency substantially.

Therefore, we adapt our RCPSP-algorithm to make use of the special structure of the SALB-1. Before we describe the algorithm and study the modifications in detail, we refer to some basic terms borrowed from project scheduling. If necessary to emphasize the difference between RCPSP and SALB-1 specific considerations we will use the term activity for the RCPSP and the term task for the SALB-1.

As for the RCPSP the search for an optimal solution is guided by the precedence tree introduced by Patterson et al. (cf. [11]). The nodes of the precedence tree correspond to the nodes of the branch-and-bound tree. The root node 1 of the tree is given by the single starting task and the leaves are copies of the only finishing task. The descendants of a node j within the precedence tree are built by the tasks that are eligible after scheduling the tasks on the path leading from the root node 1 to node j. Thereby, a task is called eligible, if all its predecessors are scheduled. Analogously to the algorithm for the RCPSP we use the set $\mathcal{ACS}_i$ to refer to the set of tasks currently scheduled up to level i. Assuming that passing the nodes j, $j=1,\ldots,i$, means scheduling the tasks g_j, $j=1,\ldots,i$, related to the nodes we obtain the set of eligible tasks on a level i, namely Y_i, as follows:

$$
\begin{aligned}
Y_1 &:= \{1\} \\
\mathcal{ACS}_1 &:= \{g_1\} = \{1\} \\
Y_{i+1} &:= Y_i \backslash \{g_i\} \cup \{k \in \mathcal{S}_{g_i}; \mathcal{P}_k \subseteq \mathcal{ACS}_i\} && i = 1,\ldots,J-1 \\
\mathcal{ACS}_{i+1} &:= \mathcal{ACS}_i \cup \{g_{i+1}\} && i = 1,\ldots,J-1
\end{aligned}
$$

In the literature the eligible tasks are sometimes called available tasks (cf. [14]). However we stay

with the term eligible for not getting confused with the resource availability. Moreover, we assume the eligible set to be sorted w.r.t. increasing labels.

For a given upper bound $\overline{T}$ on the minimal makespan of the RCPSP we obtain a bound on the latest finish (start time) of the activities by the backward pass from MPM-calculation. The calculation takes into account the upper bound by defining the latest finish times LF_J of the only finishing activity J through $LF_J = \overline{T}$. For the SALB-1 we can adapt the foregoing to derive bounds on the latest possible stations that can be assigned to the tasks (cf. e.g. Scholl and Klein [14]). Given an upper bound $\overline{V}$ on the number of stations in the optimal solution, we can derive the precedence relation based bound on the latest possible station LPS_j assignable to task j, $j = 1, \ldots, J$, without exceeding the total requirements of $\overline{V}$ stations. We use the set of successors $\overline{S}_j$ of task j, $j = 1, \ldots, J$, within the transitive closure of the network and determine the minimum workload WL_j to be processed after task j, $j = 1, \ldots, J$, is assigned to a station

$$WL_j = \sum_{l \in \overline{S}_j} t_l.$$

The bounds are then determined by

$$LPS_j = \begin{cases} \overline{V} - ((t_j + WL_j) \text{ div } c) + 1 & \text{if} \quad (t_j + WL_j) \bmod c = 0 \\ & \text{and} \quad (t_j + WL_j) \text{ div } c > 0 \\ \overline{V} - ((t_j + WL_j) \text{ div } c) + 0 & \text{otherwise} \end{cases}$$

where "div" and "mod" denote the integer division and the modulus, respectively. In Section 6 we will show how to use the SALB-1 specific bounds to reduce the latest possible station.

5.1 The Enumeration Scheme

Before we explicitly state the algorithm, we give a brief summary of the enumeration scheme: The algorithm assigns exactly one task to a station per node of the branch-and-bound tree. A task is firstly considered for assignment when it is eligible. The eligible tasks are studied in order of increasing labels. The task is assigned to the most recently opened station, if the load currently assigned to the station allows so, or, otherwise, a newly opened station. Backtracking is performed if the station that has to be assigned to a task violates the upper bound imposed by its latest possible station, or if no untested eligible task is left on the current level. On this level the next eligible task is selected. If tracking back leads to level 0 the optimal solution has been computed.

The algorithm we are going to present is an adaptation of an algorithm proposed for the RCPSP (cf. [17], [18]). The symbols and definitions used to describe the SALB-1 algorithm are given in Table 5.

i	: level index
i^*	: lowest level where a task assigned to a station violates the bound, imposed by the new latest possible station induced through adaptation after finding an improved solution
Y_i	: set of eligible tasks on level i
$t^{min}(Y_i)$	: minimum processing time of the eligible tasks of level i
$\hat{N}_i$	: cardinality of the set Y_i
N_i	: index of the element from the eligible set Y_i which is currently considered
Y_{iN_i}	: the N_i'th element of the set of eligible tasks on level i
$\mathcal{ACS}_i$ ($\overline{\mathcal{ACS}}_i$)	: set of tasks that have been assigned to a station (are unassigned) up to level i
g_i	: task currently assigned or considered on level i
v_i	: station task g_i is currently assigned to
p_i	: the position currently assigned to task g_i within station v_i
c_i	: idle time (unused cycle time) of station v_i after task g_i is assigned to it
ST_i (CT_i)	: relative start (completion) time of task g_i assigned on level i to station v_i
l_v	: lowest level on which a task is assigned to station v
FTS_v	: first task assigned to station v
LS_v	: load of station v, i.e., the set of tasks to be processed at station v
$PT(LS_v)$	: processing time of the load LS_v of station v, i.e., the sum of the processing times of the tasks assigned to station v
$IT(LS_v)$	: idle time of station v induced by the load LS_v
$\mathcal{P}(LS_v)(\mathcal{S}(LS_v))$	: set of predecessors (successors) of the tasks out of load LS_v of station v assigned to station v
WL_j	: minimum workload that has to be processed after task j is assigned to a station
LPS_j	: latest possible station assignable to task j if at most $\overline{V}$ stations are required

Table 5: Symbols and Definitions Used in the SALB-1-Algorithm

The basic enumeration scheme for solving the SALB-1 is displayed in Table 6. We contrast the SALB-1 algorithm with the RCPSP algorithm (cf. [17], [18]). Step 4, Step 7, and Step 8, are of particular interest.

Step 4 determines the station the task currently considered has to be assigned to. We contrast with Step 4 of the RCPSP algorithm. The RCPSP algorithm determines the earliest precedence and resource feasible start time. The start time is not less than the one of the most recently scheduled activity and does not exceed the latest start time. The SALB-1 algorithm determines the lowest un-fully loaded station having enough idle time for additionally processing the current task. The station is not less than the one most recently opened and does not exceed the latest possible station.

Step 1: **(Initialization)**
$\mathcal{ACS}_0 := \emptyset$; $g_0 := 0$; $v_0 = 1$; $p_0 = 0$; $c_0 := c$; $ST_0 := 0$; $CT_0 := 0$; $\mathcal{P}_1 := \{0\}$; $i := 1$; $Y_1 := \{1\}$; $\hat{N}_1 := 1$; $N_1 := 0$; $g_1 := 0$;

Step 2: **(Select next untested descendant)**
If $N_i < \hat{N}_i$ then $N_i := N_i + 1$; $g_i := Y_{iN_i}$; goto Step 4;

Step 3: **(One-level backtracking)**
$i := i - 1$; if $i = 0$ then STOP; else goto Step 2;

Step 4: **(Determine (not fully loaded) station)**
if $(t_{g_i} \leq c_{i-1})$ and $(v_{i-1} \leq LPS_{g_i})$ then $v_i := v_{i-1}$; $p_i := p_{i-1}+1$; $c_i := c_{i-1} - t_{g_i}$; $ST_i = CT_{i-1}$; $CT_i = CT_{i-1} + t_{g_i}$ $\mathcal{ACS}_i := \mathcal{ACS}_{i-1} \cup \{g_i\}$;
else if $(v_{i-1} < LPS_{g_i})$ then $v_i := v_{i-1} + 1$; $p_i = 1$; $c_i := c - t_{g_i}$; $ST_i = 0$; $CT_i = t_{g_i}$; $\mathcal{ACS}_i := \mathcal{ACS}_{i-1} \cup \{g_i\}$; else goto Step 3

Step 5: If $(i = J)$ then goto Step 7;

Step 6: **(Update the eligible set)**
$i := i + 1$; calculate the new descendant set $Y_i := Y_{i-1} \backslash \{g_{i-1}\} \cup \{k \in \mathcal{S}_{g_{i-1}}; \mathcal{P}_k \subseteq \mathcal{ACS}_{i-1}\}$; $\hat{N}_i := |Y_i|$; $N_i := 0$; goto Step 2;

Step 7: **(Store solution and adjust bounds)**
Store solution $g_j, v_j, p_j, ST_j, CT_j$, $j = 1, \ldots, J$;
Set $\overline{V} = v_J$; $LPS_j := LPS_j - (LPS_J - \overline{V} + 1)$, $j = 1, \ldots, J$;

Step 8: **(Calculate lowest indexed level producing a bound violation)**
$i^* := \min\{k \in \{1, \ldots, J\}; v_k > LPS_{g_k}\}$;

Step 9: **(Variable-stage backtracking)**
$i^* := i^* - p_{i^*} + 1$; $i = i^*$; goto Step 2;

Table 6: Basic Algorithm for the SALB-1

More precisely, if the duration t_{g_i} of task g_i currently considered at most equals the idle time c_{i-1} of the station v_{i-1} task g_{i-1} is assigned to, and, moreover, v_{i-1} is at most equal to the latest possible station LPS_{g_i} task g_{i-1} can be assigned to, then g_i is assigned to station $v_i = v_{i-1}$. Otherwise, if feasible, i.e., $v_{i-1} < LPS_{g_i}$, g_i is assigned to a newly opened station $v_i = v_{i-1} + 1$. After the task is assigned to station v_i, the idle time c_i of station v_i is computed. We additionally compute the relative start time ST_i and the relative completion time CT_i of task g_i at station v_i. The quantities will be used later for describing the bounding rules. If the task g_i currently considered cannot be assigned to a station without producing a bound violation backtracking is performed.

Step 7 stores beside the quantities memorized for the RCPSP, i.e., the (relative) start time ST_j and completion time CT_j, additionally the relative position p_j and the station v_j of task g_j, $j = 1, \ldots, J$. Moreover, the bounds imposed by the latest possible station are adapted similarly as in the RCPSP algorithm. Note, after finding an improved solution having $\overline{V} = v_J$ stations we can adapt the bounds

to guarantee that from there on only solutions employing less than $\overline{V}$ stations are generated.

Step 8 performs backtracking to the lowest level where the new bounds are violated. More precisely we can track $p_{i^*} - 1$ further steps back, since the task producing the violation has to a be assigned to a station less than v_{i^*}. The level we return to is the one where the first task is assigned to station v_{i^*}. Note, if we assume, that the stations are maximally loaded (cf. Theorem 2, Subsection 5.2) then we can track p_{i^*} levels back.

In the following subsection we will describe the search tree reduction schemes we have employed to make the basic scheme efficient. Some are borrowed from the literature and others extend them (cf. Jackson [6], Johnson [7], Scholl and Klein [14]).

5.2 Search Tree Reduction

The first rule which is presented is a preprocessing rule. It can be implemented via input data adjustment without effect on the enumeration scheme. Moreover, it can be used by any algorithm solving the problem at hand. The rule extends the task duration incrementing rule from Johnson (cf. [7]) and gains its effect with lower bounds, e.g. $LB1$, $LB4$, and $LB6$, which we will present later (cf. Section 6).

Theorem 1 *(Extended Johnson-Rule I, Extended Task Duration Incrementing, Preprocessing)*
The processing time t_g of task g, that cannot share the cycle time with any neighbor or precedence independent task $\overline{g}$, $\overline{g} \neq g$, i.e., $t_g + t_{\overline{g}} > c$ for all $\overline{g} \in (\mathcal{P}_g \cup \mathcal{S}_g) \cup (\{1,\ldots,J\} - (\overline{\mathcal{P}}_g \cup \overline{\mathcal{S}}_g))$, can be incremented to $t_g = c$ without influence on the set of feasible solutions.

For the representation of the further rules we will make use of the term sequence $Seq_i = [g_1,\ldots,g_i]$ to refer to a sequence of tasks produced by the algorithm of Table 6 by successively assigning a station to the tasks $g_1,\ldots,g_i$. If the loads of the stations $1,\ldots,v$ are considered, we denote the sequence by $Seq_i = [LS_1,\ldots,LS_v]$. It will help to recall the RCPSP formulation of the SALB-1 (cf. Section 4) when considering the search tree reduction schemes.

Note, the schedules of the RCPSP-formulation, which relate to a sequence are semi-active (cf. [19]) by construction. That is, in the corresponding RCPSP schedule the start time of none of the tasks can be reduced by one period without violation of the precedence and resource constraints, or delaying another task. For the RCPSP the following rule is a variant of the global left-shift rule (cf. [18]). It guarantees, in combination with the previous statement, that only active (cf. [19]) corresponding schedules are generated. Thereby, an RCPSP schedule is active, if the start time of no task can be reduced – at all – without violation of the precedence and resource constraints, or delaying another task.

For the SALB-1, it states that only maximally loaded stations are generated on intermediate levels.

Theorem 2 *(Jackson-Rule I, Maximum Load)*
Let $Seq_i = [g_1, \ldots, g_i]$ *be the sequence of tasks currently considered to be continued. If there is an assignable task* g, $g \in Y_{i+1}$, *with* $t_g \leq c_i$, *then none of the tasks* $\overline{g}$, $\overline{g} \in Y_{i+1}$, *with* $t_{\overline{g}} > c_i$ *has to be chosen on level* $(i+1)$ *to continue the sequence* Seq_i.

Clearly, since maximality of the load of a station depends on the tasks previously assigned, it is no absolute attribute. That is, one and the same load can be maximal within a certain sequence and not within an other.
The following rule coincides with the set-based dominance used in resource-constrained project scheduling (cf. [17]). It compares the current sequence with one previously studied. If the same tasks can be found in both sequences, and, moreover, the current sequence uses at least the same number of stations as the one analyzed earlier, then the current sequence is dominated.

Theorem 3 *(Set-Based Dominance)*
Let $Seq_i = [g_1, \ldots, g_i]$ *be the sequence currently considered to be continued. If a previously evaluated sequence* $\overline{Seq_i} = [\overline{g}_1, \ldots, \overline{g}_i]$ *has the same set of assigned tasks, i.e., (a)* $\mathcal{ACS}(Seq_i) = \{g_1, \ldots, g_i\} = \{\overline{g}_1, \ldots, \overline{g}_i\} = \mathcal{ACS}(\overline{Seq_i})$ *and (b)* $v_i \cdot c + CT_i \geq \overline{v}_i \cdot c + \overline{CT}_i$, *then the continuations of* Seq_i *are dominated by continuations of* $\overline{Seq_i}$.

Clearly, instead of the more elegant, but time consuming, description given in (b), we use the more efficient two step comparison, where, first the number of stations, and second, if necessary, the utilized processing time of the last station are compared.
The sets related to a partial assignment can be coded by the binaries of an unsigned integer (array) and stored in a binary level related tree. Using the fact that the number of bits set to 1 in an integer (array) representing a set of assigned tasks of the same level are equal, fast binary tree search can be realized.
Obviously, we would achieve the same effect with the labeling scheme developed by Schrage and Baker (cf. [15]). The scheme assigns each task a label, such that each precedence feasible subset of the set of tasks has a corresponding unique integer. The integer number is defined by the sum of the labels assigned to the elements of the set. Using the integer as an array index the precedence feasible subset can be accessed very quickly. Unfortunately, the number of precedence feasible subsets is considerably larger than the number of precedence feasible subsets to be necessarily generated.

Therefore, it seems to be more appropriate to employ the set-based dominance instead of the labeling scheme by Schrage and Baker. However, although the set-based dominance requires only a fraction of the memory used by the labeling scheme, the requirements are extremely high. That is, we need rules that perform comparison pruning without excessive use of memory.

The three solution characteristics we present in the following are partly covered by the set based dominance. However, they can easily be incorporated into the basic enumeration scheme defining an a priori reduction of the search space. Therefore, no additional effort is necessary to proof the assumptions of set based dominance. Moreover, only low memory requirements make them favorable if the number and the size of the set of assigned tasks to be stored grows too large. The first rule is implicitly used by Johnson (cf. [7]) in his enumeration scheme:

Theorem 4 *(Solution Characteristic I)*
The SALB-1 has an optimal solution where the labels of the tasks consecutively assigned to the same station are monotonically increasing, i.e., $g_i > g_{i+1}$ implies $v_i < v_{i+1}$, or equivalent, it is $g_i < g_{i+1}$ unless $v_i < v_{i+1}$.

Proof: Consider an optimal solution. Since the tasks are numerically labeled, two tasks g_{i+1} and g_{i+2}, $g_{i+1} > g_{i+2}$, assigned to the same station can be interchanged without violating the precedence constraints, exceeding the cycle time or increasing the number of stations related to the solution. □

Note, if set-based dominance pruning is employed then the memory requirements are reduced by Solution Characteristic I. In contrast, the memory requirements induced by the labeling scheme by Schrage and Baker remain unchanged, since the memory is allotted before the enumeration is started.

It is sensible to combine the modification of the enumeration scheme due to the Jackson-Rule I and the Solution Characteristic I. We abbreviate the minimum processing time of the tasks out of the eligible set Y_i to $t^{min}(Y_i)$, i.e., $t^{min}(Y_i) = \min\{t_j, j \in Y_i\}$. The combination is captured in the following remark:

Remark 1 *(Combination of Jackson-Rule I and Solution Characteristic I)*
Let $Seq_i = [g_1, \ldots, g_i]$ be the sequence currently considered to be continued. If the minimum processing time $t^{min}(Y_{i+1})$ is at most equal to unused cycle time c_i of the station v_i task g_i is assigned to, i.e., $t^{min}(Y_{i+1}) \leq c_i$, then only the tasks g, $g = Y_{i+1,N}$, $N = N_i, \ldots, \hat{N}_{i+1}$ with $t_g \leq c_i$ have to be tested on level $(i+1)$.

Proof: Let g be a task fulfilling the assumptions. If $t_{g_{i+1}} > c_i$ holds then station v_i is not maximally loaded. If $N_{i+1} < N_i$ holds then either station v_i is not maximally loaded, or the labels consecutively

assigned to station v_i are not monotonically increasing. □

The following statement transfers the idea of monotonicity of the labels of the tasks assigned to the same station to monotonicity of the first tasks assigned to different stations. The derived rule covers other portions of set-based dominance pruning and strengthens the first station dominance introduced by Johnson (cf. [7]) as well as the simple permutation rule presented by Scholl and Klein (cf. [14]). In our formulation we assume the enumeration to be reduced in accordance with Solution Characteristic I.

Theorem 5 *(Solution Characteristic II)*
Let V^* be the optimal number of stations of a SALB-1. The SALB-1 has an optimal solution where the labels of the first tasks FTS_v assigned to stations v, $v = 1, \ldots, V^*$, are monotonically increasing unless precedence constraints require the contrary. That is, for $\overline{v}$, v, $\overline{v} < v$, it is

$$(a)\ FTS_{\overline{v}} < FTS_v \quad or \quad (b)\ \mathcal{P}(LS_v) \not\subseteq LS_v \cup \mathcal{ACS}_{l_{\overline{v}}-1}.$$

Proof: Let $Seq_J = [LS_1, \ldots, LS_{V^*}]$ be an optimal solution. If condition (a) or (b) holds for the entire solution, we are done. Otherwise we select the lowest index $\overline{v}$ and related index v for which neither (a) nor (b) holds. We rearrange the sequence of workloads to the feasible sequence $\overline{Seq_J} = [LS_1, \ldots, LS_{\overline{v}-1}, LS_v, LS_{\overline{v}}, LS_{\overline{v}+1}, \ldots, LS_{v-1}, LS_{v+1}, \ldots, LS_{v^*}]$. The sequence $\overline{Seq_J}$ has, due to the ordering of Y_i, been previously studied. Repetitively applying the procedure leads to an optimal solution fulfilling the requirements. □

The rule is simply implemented. First, in Step 4, if the current decision is to close a station v with respect to sequence $Seq_i = [LS_1, \ldots, LS_v]$ for the first time, then the set of predecessors $\mathcal{P}(LS_v)$ of station load LS_v is determined, and conditions (a) and (b) are checked. If neither (a) nor (b) holds we trace back.
Note, whereas the first station dominance introduced by Johnson (cf. [7]) states, that a station load once built for the first station, need not be built for a later station, the modification by Scholl and Klein (cf. [14]) identifies two consecutive station loads that can be interchanged by using an efficient sufficient condition for interchangeability, i.e., (*) $max\{j \in LS_v\} < min\{j \in LS_{v-1}\}$. However, since interchangeability might occur although (*) is violated Solution Characteristic II has a stronger effect than the Simple Permutation Rule from Scholl and Klein.

Remark 2
Taking into account search tree reduction in accordance with Solution Characteristic II is equivalent

to employ: A load $SL_{\overline{v}}$ once built for a station $\overline{v}$ to continue sequence $Seq = [SL_1, \ldots, SL_{\overline{v}-1}]$ need not be built for a station v, $\overline{v} < v$, to continue $\overline{Seq} = [SL_1, \ldots, SL_{\overline{v}-1}, \overline{SL_{\overline{v}}}, \ldots, \overline{SL_{v-1}}]$.

Roughly speaking Solution Characteristics I and II consider the arrangement of tasks within stations and the arrangement of entire station loads. The following rule analyzes interchangeability of tasks between different stations. Again, further portions of the set-based dominance pruning are covered.

Theorem 6 *(Solution Characteristic III)*
The SALB-1 has an optimal solution where no task g_i assigned to a station v_i can be interchanged with a task g_{i-k} assigned to station v_{i-k} with $g_{i-k} > g_i$ and $v_{i-k} < v_i$ without violating the precedence constraints or exceeding the cycle time at station v_{i-k} or station v_i.

Similar to the realization of Solution Characteristic II the assumptions are checked when it is decided to close a station v_i, that is, if the station is maximally loaded. If for a task $g \in LS_{v_i}$ there is a task g_{i-k} with (a) $g_{i-k} > g$, (b) $v_{i-k} < v_i$, (c) $IT(LS_{v_i})) + t_g \geq t_{g_{i-k}}$, (d) $IT(LSv_{g_{i-k}}) + t_{g_{i-k}} \geq t_g$, (e) $\mathcal{P}_g \subseteq \mathcal{ACS}_{i-k-1}$, (f) $\mathcal{S}_{g_{i-k}} \cap \mathcal{ACS}_{i-p_i} = \emptyset$, then we can trace back. Conditions (a) and (b) indicate a violation of the monotonicity of task labels, (c) and (d) ensure that the cycle time is not exceeded if the task g_{i-k} and task g_i are interchanged, (e) and (f) preserve precedence feasibility.
Note, if the tasks are labeled with respect to the processing times, such that $i < j$ implies $t_i \geq t_j$ or $i \in \mathcal{P}_j$ then the Extended Jackson rule provided by Scholl and Klein is enhanced (cf. [14]).
Clearly, one can extend the considerations to interchangeability of subsets of station loads $SSL_{\overline{v}} \subseteq SL_{\overline{v}}$ and $SSL_v \subseteq SL_v$, $\overline{v} < v$, with $min\{j \in SSL_{\overline{v}}\} > min\{j \in SSL_v\}$. However, serious effort is necessary to find the subsets and verify (the adapted) assumptions.

6 Lower Bounds for the SALB-1

In this section we will summarize the lower bounds developed for the SALB-1. We strengthen the effect and reduce the complexity of some of them. For a more thorough discussion we refer to [13]. For convenience we denote the set of tasks by $\mathcal{J}$.
The **first bound** $LB1$ is derived by neglecting the precedence relations, and, moreover, assuming that all the stations but at most one are fully loaded. We use $\lceil x \rceil$ to denote the lowest integer that is at least equal to x and determine

$$LB1(\mathcal{J}) = \sum_{j \in \mathcal{J}}^{J} t_j \text{ div } c + \min\{\sum_{j \in \mathcal{J}} t_j \bmod c, 1\} = \left\lceil \sum_{j \in \mathcal{J}} t_j / c \right\rceil = \left\lceil \overline{LB1}(\mathcal{J}) \right\rceil .$$

By building the sum of processing times before the enumeration is started, and decrementing the sum after the assignment of a task the bound can be realized with constant effort.

The **second bound** $LB2$ and the third bound $LB3$ have been introduced by Johnson (cf. [7]). Both bounds relax the precedence relations too, and, thus, can be used for bin packing problems as well. They key idea is to use portions of the cycle time to measure the processing times of the tasks instead of the number periods. Doing so, a task can, e.g., have a duration which is (a) less than a half of the cycle time, (b) equal to a half of the cycle time, or (c) larger than a half of a cycle time. Given the classification of the durations, the tasks from (a) are excluded from further consideration. The tasks from (b) can only share a station with another single task of (b), and the tasks of (c) have to be assigned to a station on their own. As introduced by Scholl (cf. [13], pp. 43), we use the interval notation $\mathcal{J}\left[0c, \frac{1}{2}c\right)$, $\mathcal{J}\left[\frac{1}{2}c, \frac{1}{2}c\right]$, $\mathcal{J}\left(\frac{1}{2}c, \frac{1}{1}c\right]$, to refer to the set of tasks out of $\mathcal{J}$, the processing times of which belong to the intervals $\left[0c, \frac{1}{2}c\right)$, $\left[\frac{1}{2}c, \frac{1}{2}c\right]$, $\left(\frac{1}{2}c, \frac{1}{1}c\right]$, and obtain

$$LB2(\mathcal{J}) = \left\lceil \frac{1}{2} \cdot \left| \mathcal{J}\left[\frac{1}{2}c; \frac{1}{2}c\right] \right| + \frac{1}{1} \cdot \left| \mathcal{J}\left(\frac{1}{2}c; \frac{1}{1}c\right] \right| \right\rceil = \left\lceil \overline{LB2}(\mathcal{J}) \right\rceil .$$

The **third bound** $LB3$ relies on the idea of measuring the durations in thirds of cycle time. With similar arguments as above we obtain

$$\begin{aligned} LB3(\mathcal{J}) &= \left\lceil \frac{1}{3} \cdot \left| \mathcal{J}\left[\frac{1}{3}c; \frac{1}{3}c\right] \right| + \frac{1}{2} \cdot \left| \mathcal{J}\left(\frac{1}{3}c; \frac{2}{3}c\right) \right| + \frac{2}{3} \cdot \left| \mathcal{J}\left[\frac{2}{3}c; \frac{2}{3}c\right] \right| + \frac{1}{1} \cdot \left| \mathcal{J}\left(\frac{2}{3}c; \frac{1}{1}c\right] \right| \right\rceil \\ &= \left\lceil \overline{LB3}(\mathcal{J}) \right\rceil . \end{aligned}$$

A general formula when measuring the duration in fractions $1/h$, $h = 2, \ldots, \lceil \frac{J}{\overline{LB1}(\mathcal{J})} \rceil$, of the cycle time has been developed by Johnson (cf. [7]).

Note, as mentioned by Scholl (cf. [13], p. 44), the bound is of complexity $\mathcal{O}(|\mathcal{J}|)$. However, if used during the enumeration, the effort necessary can be reduced substantially. Before the enumeration starts one (1) classifies the tasks in accordance with the intervals specified above, and (2) determines the power of the sets. During the enumeration, (1) if a task is assigned to a station, the power of the sets it belongs to is decremented, (2) if a task is removed from a station, the power of the set the task belongs to is incremented. Consequently, only constant effort is necessary to evaluate the formulae during enumeration.

The **fourth bound** $LB4$ is Scholl's (cf. [13], pp. 44) extension of Bound Argument 4 from Johnson ([7]). The idea is to relax the SALB-1 to a single machine scheduling problem. We summarize the outline from Scholl. The tasks are considered as jobs that have to be processed consecutively on a single machine. The processing time of job j is $p_j = t_j/c$, $j = 1, \ldots, J$. After processing a job j on the machine a certain amount of time has to pass before the job can be considered as finished. The

amount of time is referred to as tail of job j and denoted as n_j. Again the objective is to minimize the makespan. Given that the tasks are labeled in order of non-increasing tails, i.e., $n_i \geq n_{i+1}$, $i = 1, \ldots, |\mathcal{J}| - 1$, the minimal makespan is $LB4(\mathcal{J}) = \lceil \max_{k=1}^{|\mathcal{J}|} \{n_k + \sum_{i=1}^{k} p_i\} \rceil = \lceil \overline{LB4}(\mathcal{J}) \rceil$. We compute the tails similar to Scholl. Adding the fact that the tails can be rounded to the next integer – which is expressed by the first case of the following distinctions – if the task the successors of which are studied cannot share a station with any of its successors, we obtain

$$n_{j1} = \begin{cases} LB1(\overline{\mathcal{S}}_j) \text{ , if } p_j + p_s > 1 \text{ for} \\ \qquad\qquad \text{all } s \in \mathcal{S}_j \\ \overline{LB1}(\overline{\mathcal{S}}_j) \text{ , otherwise} \end{cases} \qquad n_{j4} = \begin{cases} LB4(\overline{\mathcal{S}}_j) \text{ , if } p_j + p_s > 1 \text{ for} \\ \qquad\qquad \text{all } s \in \mathcal{S}_j \\ \overline{LB4}(\overline{\mathcal{S}}_j) \text{ , otherwise} \end{cases}$$

$$n_{j2} = \begin{cases} LB2(\overline{\mathcal{S}}_j) \quad \text{, if } p_j + p_s > 1 \text{ for} \\ \qquad\qquad \text{all } s \in \mathcal{S}_j \\ \overline{LB2}(\overline{\mathcal{S}}_j) \quad \text{, else if } p_j \geq \frac{1}{2} \text{ or} \\ \qquad\qquad LB2(\overline{\mathcal{S}}_j) \notin \mathbf{N} \\ \overline{LB2}(\overline{\mathcal{S}}_j) - \frac{1}{2} \text{ , otherwise} \end{cases} \qquad n_{j3} = \begin{cases} LB3(\overline{\mathcal{S}}_j) \quad \text{, if } p_j + p_s > 1 \text{ for} \\ \qquad\qquad \text{all } s \in \mathcal{S}_j \\ \overline{LB3}(\overline{\mathcal{S}}_j) \quad \text{, else if } p_j \geq \frac{2}{3} \\ \\ \overline{LB3}(\overline{\mathcal{S}}_j) - \frac{1}{3} \text{ , otherwise} \end{cases}$$

Taking into account that the network has a dummy starting and a dummy finishing task, we can recursively calculate the tails as proposed by Scholl (cf. [13], page 46). Analogously, by replacing the (transitive) successors through the (transitive) predecessors, we can compute the heads a_j, $j = 1, \ldots, J$. Using the heads and tails we obtain a lower bound for the single machine scheduling problem with heads and tails $Z = \max_{j=1}^{J} \{a_j + p_j + n_j\}$. This bound is then used as Scholl's (cf. [13], page 48) generalized bound $LB4'$.

Now, for a given bound $\overline{V}$, we can employ the tails to recalculate the latest possible station LPS_j, derived in Section 5, a task j can be assigned to. It is

$$LPS_j(\overline{V}) := \begin{cases} \overline{V} & \text{, if } n_j = 0 \\ \overline{V} + 1 - \lceil p_j + n_j \rceil & \text{, otherwise} \end{cases} \qquad j = 1, \ldots, J \qquad (11)$$

In addition to Scholl and Klein ([14]), who used the bound only for the root problem, we implicitly utilized the results of the computation of $LB4'$ as given in (11), and thus, at least partly, use $LB4'$ during enumeration. More precisely, before a task g_i is assigned to a station v_i, it is checked if the station number exceeds the currently valid latest possible station, i.e., if $v_i > LPS_{g_i}$, then, assuming maximally loaded stations, we can track back to the level were the last task is assigned to station $v_i - 1$.

The **fifth bound** $LB5$ (cf. [14]) additionally needs the earliest possible station which is determined

as follows

$$EPS_j := \begin{cases} 1 & \text{, if } a_j = 0 \\ \lceil a_j + p_j \rceil & \text{, otherwise} \end{cases} \qquad j = 1, \ldots, J. \tag{12}$$

We obtain

$$LB5 = \min\{V \geq 0; LPS_j(V) \geq EPS_j, j = 1, \ldots, J\}$$

The **sixth bound** $LB6$ has been developed by Berger et al. (cf. [2]). It can be described in five steps. In step 1, the tasks are sorted with respect to non-increasing durations. In step 2, the tasks are grouped in the classes $\mathcal{J}\left(0; \frac{1}{3}c\right]$, $\mathcal{J}\left(\frac{1}{3}c; \frac{1}{2}c\right]$, $\mathcal{J}\left(\frac{1}{2}c; \frac{1}{1}c\right]$. In step 3, beginning with the longest task, the tasks out of $\mathcal{J}\left(\frac{1}{2}c; \frac{1}{1}c\right]$ are assigned to separate stations. In step 4, beginning with the shortest task, the tasks out of $\mathcal{J}\left(\frac{1}{3}c; \frac{1}{2}c\right]$ are tried to be assigned to the stations opened up to now. The stations are considered in order of non-decreasing idle times. The tasks that can be assigned are named $\mathcal{J}^{ass}\left(\frac{1}{3}c; \frac{1}{2}c\right]$, the remaining tasks $\mathcal{J}^{unass}\left(\frac{1}{3}c; \frac{1}{2}c\right]$ out of $\mathcal{J}\left(\frac{1}{3}c; \frac{1}{2}c\right]$ are too much or too long to fit in one of the currently opened station. But, two of them can share a newly opened station, and we get a preliminary bound

$$PLB = \left|\mathcal{J}\left(\frac{1}{2}c; \frac{1}{1}c\right]\right| + \left\lceil \frac{1}{2}\left|\mathcal{J}^{unass}\left(\frac{1}{3}c; \frac{1}{2}c\right]\right| \right\rceil.$$

The preliminary bound can be improved. In step 5, we consider the integers s out of the interval $\left[0; \frac{1}{3}c\right)$. For any of these integers a task j, $j \in \mathcal{J}\left(s, \frac{1}{3}c\right]$, can only be added to a station related to tasks out of $\mathcal{J}^{unass}\left(\frac{1}{3}c; \frac{1}{2}c\right] \cup \mathcal{J}\left(\frac{1}{2}c; c - s\right]$. The idle time of these stations is considered. If the idle time suffices to schedule the tasks out of $\mathcal{J}\left(s, \frac{1}{3}c\right]$, then the prelimiary bound PLB remains unchanged, otherwise the bound is increased by appying $LB1$ to the remaining overhead. Considering all the s, $s \in [0, \frac{1}{3}c)$, we use the maximum increment to improve the preliminary bound PLB to $LB6(\mathcal{J})$.

The **seventh bound** $LB7$ has been developed by Scholl (cf. [13]) who considers the problem of minimizing the cycle time for a given number of stations. For an initially given number of stations $\overline{V}$ a lower bound $\overline{c}(\overline{V})$ for the cycle time is determined. If the lower bound $\overline{c}(\overline{V})$ exceeds the actual cycle time, then $\overline{V}$ is incremented by one, and the procedure is repeated, otherwise the current $\overline{V}$ is a lower bound. Scholl determined the bound $\overline{c}(\overline{V})$ as follows: First, all the tasks are labeled with respect to non-increasing order, i.e., $t_i \geq t_{i+1}$, $i = 1, \ldots, n-1$. Second, assuming that $J > \overline{V}$ tasks have to be assigned, a lower bound on the cycle time is $t_{\overline{V}} + t_{\overline{V}+1}$; assuming that $J > 2 \cdot \overline{V}$ tasks have to be assigned a lower bound on the cycle time is $t_{2\overline{V}-1} + t_{2\overline{V}} + t_{2\overline{V}+1}$, and so on. In general we obtain,

$$\overline{c}(\overline{V}) = max\{\sum_{i=0}^{k} t_{k \cdot \overline{V}+1-i}; k = 1, \ldots, \lfloor (J-1)/\overline{V} \rfloor\}$$

This bound has only been used for the root-problem. As initial value for $\overline{V}$ we employed the maximum of the bounds $LB1, \ldots, LB6$.

7 Computational Results

The algorithm has been coded in GNU C and implemented on a personal computer (66 MHz, 80486 dx) under the Linux operating system. From the bounds described in Section 6 we applied the bound

$$LB = \max\{LB_1, LB_2, LB_3, LB_4', LB_5, LB_6, LB_7\}$$

to the root problem. The bounds LB_1, LB_2 and LB_3 have been used during the enumeration whenever a station is decided to be closed because it is maximally loaded. The bound LB_4 has been implicitly utilized through the adaptation of the latest possible station after finding an improved solution. Clearly, before determining the bounds we have applied the extended task incrementing rule. Moreover, the backtrack level after finding an improved solution with $\overline{V}$ stations can be reduced further. We assume that the latest possible station LPS_J of task J has been adapted, define

$$LBX(\overline{ACS}_{l_v-1}) \;:=\; max\{LB1(\overline{ACS}_{l_v-1}), LB2(\overline{ACS}_{l_v-1}), LB3(\overline{ACS}_{l_v-1})\} \quad v = 1 \ldots \overline{V},$$

and obtain

$$\bar{i} \;=\; min\{l_v - 1, (v-1) + LBX(\overline{ACS}_{l_v-1}) > LPS_J,\; v = 1 \ldots \overline{V}\}$$

as an (improved) variable backtrack level. Furthermore, the enumeration scheme employs all the search tree reduction schemes presented in Subsection 5.2, but the Set-Based Dominance (cf. Theorem 3). Instead Solution Characteristic I, II and III are utilized to exclude dominated assignments from further continuation. Obviously, doing so, reduces the memory requirements substantially.

Before the enumeration has been started the tasks have been relabeled with the respect to the maximum duration rule (cf. Subsection 5.2). As first and second tie-breaker we used the minimum latest possible station and the minimum task label, respectively.

The enumeration (simultaneously) considers the (relabeled) original and the (relabeled) reversed problem. The reversed problem is obtained from the original problem through the reversal of the precedence relations, i.e., task j becomes task $J-j+1$, $j = 1, \ldots, J$, in the reversed problem and a predecessor h, $h \in \mathcal{P}_j$, becomes a successor of task j in the reversed problem. Afterwards the tasks are relabeled in accordance with the priority rule. The enumeration switches every 500.000 node evaluations from the examination of the original problem to the examination of the reversed problem, and back after further 500.000 node evaluations.

Set	Combined			Talbot et al.			Hoffmann			Scholl		
no prob.	269			64			50			168		
Alg.	FAB	EUR	SAL	FAB	EUR	SAL	FAB	EUR	SAL	FAB	EUR	SAL
opt.	179	194	224	64	64	64	48	47	48	80	96	125
$\overline{\Delta}$	0.95	0.72	0.46	0.00	0.00	0.00	0.25	0.34	0.25	1.44	1.05	0.67
Δ^{max}	7.69	12.00	7.69	0.00	0.00	0.00	7.69	7.69	7.69	7.14	12.00	4.55
$\overline{cpu}[sec.]$	175.9	199.1	98.6	0.2	5.3	0.2	48.8	78.4	40.4	267.1	293.9	145.9
Alg.	Adapted General Sequencing Algorithm (AGSA)											
opt.	243			64			48			144		
$\overline{\Delta}$	0.25			0.00			0.25			0.33		
Δ^{max}	7.69			0.00			7.69			4.55		
$\overline{cpu}[sec.]$	65.9			0.2			24.8			98.1		

Table 7: Computational Results with Time Limit 500 Seconds

The computational results are compared with Scholl and Kleins (cf. [14]) implementation of EUREKA (cf. [5]), FABLE (cf. [7]), and SALOME (cf. [14]), where SALOME is the best algorithm currently available. All the algorithms have been implemented by Scholl in Borland Pascal 7.0 under the DOS operating system and run on a personal computer (80486 dx, 66 Mhz).

The comparison is displayed in Table 7 and Table 8. Table 7 reveals the results on different problems sets, the Talbot-set (64 instances), the Hoffmann-set (50 instances), and the Scholl-set (168 instances). Due to duplications of several instances the combined set has only 269 instances (cf. [12]). The number of tasks range between 8 and 297. For a given time limit of 500 seconds, the number of problems that have been optimally solved (opt.), the average deviations $\overline{\Delta}$, the maximum deviations Δ^{max}, and the average CPU-time $\overline{cpu}[sec.]$ are displayed. The average deviations are determined through the deviation from optimum – where known, and the lower bound otherwise. The results show that the adapted general sequencing algorithm (AGSA) can compete quite well with the (special purpose) developments FABLE, EUREKA and SALOME. On the combined set AGSA solves more instances, produces a lower average deviation from the optimum, and requires less CPU-time than the other algorithms. Three instances out of the Scholl-set have been solved for the first time.

Table 8 shows the number of problems optimally solved, the average deviation, the maximum deviation, and the average CPU-time for time limits of 50, 100, 250, and 1000 seconds. Within all the time limits the results of AGSA compare quite well with the ones of FABLE, EUREKA and SALOME. Note, AGSA solves within a time limit of 100 seconds more problems, i.e. 230, than the best algorithm

Time Limit	50 sec.			100 sec.			250 sec.			1000 sec.		
Alg.	FAB	EUR	SAL	FAB	EUR	SAL	FAB	EUR	SAL	FAB	EUR	SAL
opt.	169	183	209	170	187	213	177	190	216	181	199	227
$\overline{\Delta}$	1.10	0.81	0.61	1.07	0.76	0.58	0.98	0.76	0.54	0.91	0.64	0.43
Δ^{max}	8.82	12.00	7.69	8.82	12.00	7.69	7.69	12.00	7.69	7.69	12.00	7.69
$\overline{cpu}[sec.]$	19.7	23.5	13.0	38.2	44.4	23.8	91.5	103.6	54.2	340.8	377.6	177.4
Alg.	Adapted General Sequencing Algorithm (AGSA)											
opt.	221			230			238			247		
$\overline{\Delta}$	0.46			0.38			0.29			0.23		
Δ^{max}	7.69			7.69			7.69			7.69		
$\overline{cpu}[sec.]$	12.6			20.3			39.4			107.9		

Table 8: Computational Results with Different Time Limits

currently available – SALOME – can solve within 1000 seconds, i.e., 227. Moreover, the average deviation within 100 seconds for AGSA, 0.38%, is less than the one produced by SALOME within 1000 seconds, 0.43%. This should balance the different platforms and show the competitiveness of AGSA.

The problems that could not be solved to optimality within 1000.00 seconds have been proposed by Arcus (set 2), Bartholdi (set 2), Lutz (set 2), Mukherjee, Scholl, and Wee/Magazine (cf. [14]). The assembly projects consist of 111, 148, 89, 94, 297, and 75 tasks.

8 Conclusions

We have presented an adapted version of a general algorithm for solving the simple assembly line balancing problem of type-1. The algorithm is guided by the precedence tree, which has successfully employed to solve a wide range of resource-constrained project project scheduling problems.

Through the transparency of the precedence tree guided scheme classical dominance concepts provided by Jackson and Johnson can be enhanced substantially. The solution characteristics provided allow to substitute memory intensive set-based dominance pruning through the application of low memory requiring characteristics. The computational results show that the algorithm can compete with the best algorithms currently available for solving the SALB-1.

The results indicate that it could be of interest to study the effect of other adaptations for solving further variants of the assembly line balancing problem. Moreover, since, in contrast to claims made

in the literature, there are several problems which have less than 100 tasks that could not be solved to optimality by neither AGSA nor SALOME, it is useful to know the limits of the best solution procedure available so far. A generator of instances similar to ProGen (cf. [9]) could help to find the limits. Moreover, a proper set of characterized projects can help to study the effect of search tree reduction schemes and thus accelerate the development. Additionally, the sets would help to make algorithms more comparable even if different platforms are used.

Acknowledgments: I am grateful to Armin Scholl who provided the instances as well as his source codes of EUREKA, FABLE and SALOME.

References

[1] BAYBARS, I. (1986): A survey of exact algorithms for the simple assembly line balancing problem. Management Science, Vol. 32, pp. 909–932.

[2] BERGER, H.; J.-M. BOURJOLLY; G. LAPORTE (1992): Branch-and-bound algorithms for the multi-product assembly line balancing problem. European Journal of Operational Research, Vol. 58, pp. 215–222.

[3] BOWMAN, E.H. (1960): Assembly line balancing by linear programming. Operations Research, Vol. 8, pp. 385–389.

[4] DREXL, A. (1990): Fließbandaustaktung, Maschinenbelegung und Kapazitätsplanung in Netzwerken – Ein integrierender Ansatz. Zeitschrift für Betriebswirtschaft, Jg. 60, pp. 53–70.

[5] HOFFMANN, T. R. (1992): EUREKA: A hybrid system for assembly line balancing. Management Science. Management Science, Vol. 38, pp. 39–47.

[6] JACKSON, J.R. (1956): A computing procedure for a line balancing problem. Management Science, Vol. 2, pp. 261–271.

[7] JOHNSON, R.V. (1988): Optimally balancing large assembly lines with 'FABLE'. Management Science, Vol. 34, pp. 240–253.

[8] KLEIN, R. AND A. SCHOLL (1996): Maximizing the production rate in simple assembly line balancing - A branch and bound procedure. European Journal of Operational Research, Vol. 91, pp. 367–385.

[9] KOLISCH, R.; A. SPRECHER AND A. DREXL (1995): Characterization and generation of a general class of resource-constrained project scheduling problems. Management Science, Vol. 41, pp. 1693–1703.

[10] PATTERSON, J.H. AND J.J. ALBRACHT (1975): Assembly-line balancing: Zero-one programming with Fibonacci search. Operations Research, Vol. 23, pp. 166–172.

[11] PATTERSON, J.H.; R. SLOWINSKI; F.B. TALBOT AND J. WEGLARZ (1989): An algorithm for a general class of precedence and resource constrained scheduling problems. In: Slowinski, R. and J. Weglarz (Eds.): Advances in project scheduling. Elsevier, Amsterdam, pp. 3–28.

[12] SCHOLL, A. (1993): Data of assembly line balancing problems. Research Report, No. 16/93, Technische Hochschule Darmstadt, Germany.

[13] SCHOLL, A. (1995): Balancing and Sequencing of Assembly Lines. Physica-Verlag, Heidelberg.

[14] SCHOLL, A. AND R. KLEIN (1996): SALOME: A bidirectional branch and bound procedure for assembly line balancing. To appear in: INFORMS Journal on Computing.

[15] SCHRAGE, L. AND K.R. BAKER (1978): Dynamic programming of sequencing problems with precedence constraints. Operations Research, Vol. 26, pp. 444–459.

[16] SPRECHER, A. (1994): Resource-constrained project scheduling: Exact methods for the multi-mode case. Lecture Notes in Economics and Mathematical Systems, No. 409, Springer, Berlin.

[17] SPRECHER, A. (1996): Solving the RCPSP Efficiently at Modest Memory Requirements. Manuskripte aus den Instituten für Betriebswirtschaftslehre, No. 425, Kiel.

[18] SPRECHER, A. AND A. DREXL (1996): Solving Multi-Mode Resource-Constrained Project Scheduling Problems by a Simple, General and Powerful Sequencing Algorithm. To appear in: European Journal of Operational Research.

[19] SPRECHER, A.; R. KOLISCH AND A. DREXL (1995): Semi-active, active and non-delay schedules for the resource-constrained project scheduling problem. European Journal of Operational Research, Vol. 80, pp. 94–102.

[20] TALBOT, F.B. (1982): Resource-constrained project scheduling with time-resource tradeoffs: The non-preemptive case. Management Science, Vol. 28, pp. 1197–1210.

[21] TALBOT, F.B. AND J.H. PATTERSON (1978): An efficient integer programming algorithm with network cuts for solving resource-constrained scheduling problems. Management Science, Vol. 24, pp. 1163–1174.

[22] WHITE, W.W. (1961): Comments on a paper by Bowman. Operations Research, Vol.9, pp. 274–276