This article was downloaded by: [141.161.91.14] On: 05 September 2014, At: 09:39
Publisher: Institute for Operations Research and the Management Sciences (INFORMS)
INFORMS is located in Maryland, USA

Management Science

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

A Survey of Exact Algorithms for the Simple Assembly Line Balancing Problem

İlker Baybars,

To cite this article:
İlker Baybars, (1986) A Survey of Exact Algorithms for the Simple Assembly Line Balancing Problem. Management Science 32(8):909-932. http://dx.doi.org/10.1287/mnsc.32.8.909

Full terms and conditions of use: http://pubsonline.informs.org/page/terms-and-conditions

This article may be used only for the purposes of research, teaching, and/or private study. Commercial use or systematic downloading (by robots or other automatic processes) is prohibited without explicit Publisher approval, unless otherwise noted. For more information, contact permissions@informs.org.

The Publisher does not warrant or guarantee the article's accuracy, completeness, merchantability, fitness for a particular purpose, or non-infringement. Descriptions of, or references to, products or publications, or inclusion of an advertisement in this article, neither constitutes nor implies a guarantee, endorsement, or support of claims made of that product, publication, or service.

© 1986 INFORMS

Please scroll down for article—it is on subsequent pages

INFORMS is the largest professional society in the world for professionals in the fields of operations research, management science, and analytics.
For more information on INFORMS, its publications, membership, or meetings visit http://www.informs.org

Downloaded from informs.org by [141.161.91.14] on 05 September 2014, at 09:39 . For personal use only, all rights reserved.

MANAGEMENT SCIENCE
Vol. 32, No. 8, August 1986
Printed in U.S.A.

A SURVEY OF EXACT ALGORITHMS FOR THE SIMPLE ASSEMBLY LINE BALANCING PROBLEM*

İLKER BAYBARS
Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213

In this survey paper we discuss the development of the *simple assembly line balancing problem* (SALBP); modifications and generalizations over time; present alternate 0-1 programming formulations and a general integer programming formulation of the problem; discuss other well-known problems related to SALBP; describe and comment on a number of *exact* (i.e., optimum-seeking) methods; and present a summary of the reported computational experiences. All models discussed here are *deterministic* (i.e., all input parameters are assumed to be known with certainty) and all the algorithms discussed are exact. The problem is termed "simple" in the sense that no "mixed-models", "subassembly lines", "zoning restrictions", etc. are considered.

Due to the richness of the literature, we exclude from discussion here (a) the inexact (i.e., heuristic/approximate) algorithms for SALPB and (b) the algorithms for the general assembly line balancing problem (including the stochastic models).

(PRODUCTION/SCHEDULING—LINE BALANCING; NETWORKS/GRAPHS—APPLICATIONS; PROGRAMMING—INTEGER ALGORITHMS)

1. Introduction

1.1. *Assembly Lines*

An *assembly system* performs a set of distinct *minimum rational work elements* (*task,* hereafter) for the assembly of a product(s) and it consists of a set of *work stations* (*station,* hereafter) linked together by a transport mechanism and a detailed specification of how the assembly of the product flows from one station to another. A *task* is a smallest indivisible work element and a *station* is a location along the flow line where the tasks are processed and it consists of human/robotic operators and/or machinery/equipment (by definition, a task cannot be divided between two or more stations without conflict). The series of stations and the transport mechanism, usually a conveyor, is referred to as the *assembly line.* A manufacturing item is fed to the first station of the line at a predetermined constant *feed rate.* The conveyor moves at each interval of T time units, known as the *cycle time.* The time required for the completion of a task is termed the *process time.* The sum of the process times of all tasks assigned to a station for processing there is known as the *work content* of that station. Since the cycle time is T, the item is available to a station for only T time units, and, therefore, the work content of a station should not exceed T, so that the line can operate smoothly with no delays. The *production rate,* or, equivalently, the *feed rate,* for the system is $1/T$, that is, one unit of the finished product emerges from the last station along the line every T time units. The cycle time is traditionally predetermined, based on the demand for the product in the given period and/or the given operating time for the manufacturing system in that period. But tasks cannot be assigned to the stations arbitrarily because of technological sequencing requirements, known as *precedence relations;* the processing of a task may not start until certain tasks, *i.e., its immediate predecessors* have been processed. The precedence relations are represented schematically by a *precedence network/diagram* whose nodes correspond to tasks and if task i is an immediate predecessor of task j (i.e., if the processing of task j cannot

* Accepted by David G. Dannenbring; received April 9, 1984. This paper has been with the author 4 months for 3 revisions.

0025-1909/86/3208/0909$01.25
Copyright © 1986, The Institute of Management Sciences

start until after the completion of task *i*), this relation is represented by a directed arc (*i*, *j*) in the precedence network/diagram, joining node *i* to node *j*. The set of precedence relations is simply a *partial ordering* of the tasks.

The assembly line is said to be *balanced* if total slack (i.e., the sum of the idle times of all the stations along the line) is as low as possible. For a fixed cycle time, this can be attained by minimizing the number of stations (§3.1). If the tasks can be grouped so that all the station work contents are exactly equal, the line is said to have *perfect balance*. In most practical situations it is very difficult to achieve perfect balance. (Note: Even though we have described here just one physical representation, unpaced lines and lines which move at a constant pace are equally well represented by the discussion here and the formulations that will follow. We should, however, distinguish between the 'assembly line' referred to here versus a 'transfer line': the latter is an automated flow line with automatic material handling from station to station and with specialized task processing. In transfer lines tasks performed are often restricted by fixed machine cycles (Groover 1980), unlike in the assembly lines where the main restriction is the production rate.)

1.2. *Overview*

In this paper we discuss only *deterministic* models (i.e., all input parameters are given, and assumed to be known with certainty) and *exact* (i.e., optimum seeking) algorithms. Because there exists a large number of models and methodologies, we exclude from this survey the stochastic models and algorithms (e.g., Sphicas and Silverman 1976, Vrat 1976 and Kao 1979). Also excluded are the inexact (i.e., heuristic/approximate) methods (e.g., Tonge 1961, Helgeson and Birnie 1961, Arcus 1966, Kilbridge and Wester 1962, Dar-El 1973 and Pinto, Dannenbring and Khumawala 1978). Inexact methods are discussed separately in Baybars (1984) and Baybars and Frieze (1986).

In §2 we define the simple line balancing problem and discuss its variations and generalizations and the related scheduling/sequencing problems. This is followed by mathematical formulations of the two better known versions of the problem in §3, as well as a discussion of the complexity of these problems and measures for complexity. In §4 we first discuss the exact algorithms for the "type-1" problem in three main groups. We then present a summary of reported computational experience and make some remarks on these procedures. §5 is similar but for the "type-2" problem. Finally, in §6, we point out some issues pertaining to these problems that need to be studied further.

2. The Deterministic Assembly Line Balancing Problem

The following are true for *all* the models discussed in this paper, unless otherwise stated:

(A-1): all input parameters are known with certainty.

(A-2): a task cannot be split among two or more stations.

(A-3): tasks cannot be processed in arbitrary sequences due to technological precedence requirements.

(A-4): all tasks must be processed.

The *assembly line balancing problem* (ALBP) is one of assigning the tasks to the stations while optimizing some criterion and not violating a number of possible restrictions. Various optimization criteria and possible restrictions/requirements will be discussed below.

2.1. *The Simple Assembly Line Balancing Problems* (*SALBP*)

In addition to (A-1)–(A-4) above, the following are true for SALBP:

(A-5): all stations under consideration are equipped and manned to process any

Downloaded from informs.org by [141.161.91.14] on 05 September 2014, at 09:39 . For personal use only, all rights reserved.

one of the tasks (i.e., it is assumed, in effect, that the fixed and variable costs associated with all the stations are the same and, therefore, they need not be considered in the model).

(A-6): the task process times are independent of the station at which they are performed and of the preceding or following tasks (i.e., process times are fixed and, furthermore, they are not sequence dependent).

(A-7): any task can be processed at any station (i.e., there are no positional, layout or zoning restrictions).

(A-8): the total line is considered to be serial with no feeder or parallel subassembly lines (and, therefore, process times are additive at any station) or any possible interaction of this type is ignored.

(A-9): the assembly system is assumed to be designed for a unique model of a single product.

We can now define the first version, namely SALBP-1, of the SALBP: In addition to (A-1)–(A-9), we have:

(A-10): the cycle time T is given and fixed.

The *goal* is to *minimize total slack* which is equivalent to minimizing the number of stations along the line (see §3.1). *SALBP*-1 is also referred to as the "line balancing problem" (e.g., Jackson 1956 and Freeman 1968) or the "single-model ALBP" (e.g., Dar-El 1975 and Pinto, Dannenbring and Khumawala 1978) or "type-1 ALBP" (e.g., Mastor 1970 and Wee and Magazine 1981a) or the "basic ALBP" (e.g., Johnson 1983).

The second version of the problem, namely SALBP-2, is the same as SALBP-1 except that instead of (A-10) we have:

(A-11): the number of stations is given and fixed.

The *goal* is to *minimize the cycle time* or equivalently, to maximize the production rate. *SALBP*-2 is also known as the "type-2 ALBP" (e.g., Mastor 1970 and Wee and Magazine 1981b).

2.2. *The General Assembly Line Balancing Problem* (*GALBP*)

ALBP has been defined in many different ways by relaxing one, or any combination, of the assumptions (A-1)–(A-9), with the exception of (A-5) which will be discussed shortly. The line may be used for the production of two or more models of the same product in batches, known as the *multi-model* case or the line may be used for the production of two or more models of the same product, not in batches but they may be intermixed, known as the *mixed-models* case (e.g., Thomopoulos 1967, Dar-El 1978, Macaskill 1972 and Dar-El and Cother 1975). Furthermore, there may exist *zoning constraints* (e.g., Mitchell 1957 and Tonge 1960) restricting the grouping of certain tasks at the same station/area; there may exist *must-do-tasks* that have to be processed at a particular station (e.g., Gunther, Johnson and Peterson 1983 and Baybars 1985); there may be restrictions on *balance delay* (e.g., Kilbridge and Wester 1961), namely, the amount of idle time on the line due to unequal task assignments to the stations; there may exist *parallel stations* (e.g., Tonge 1961 and Pinto, Dannenbring and Khumawala 1975); there may exist other forms of *positional restrictions;* buffer stocks and other generalities such as *feeder* or *parallel subassembly lines* (e.g., Nanda and Scher 1976); other extensions and generalities in Mansoor (1964b) and Freeman (1967); and a goal programming approach in Gunther, Johnson and Peterson (1983). An excellent discussion of these can be found in Groover (1980) and solution methods for certain cases in Johnson (1983). Whether the goal is to minimize total slack or to minimize the number of the stations along the line, these problems will be referred to as the *general assembly line balancing problem.* Thus, GALBP is a generalization of SALBP-1 and SALBP-2. Due to the richness of the literature the GALBP models and algorithms are beyond the scope of this survey.

Downloaded from informs.org by [141.161.91.14] on 05 September 2014, at 09:39 . For personal use only, all rights reserved.

In GALBP there is no explicit concern for the (fixed) cost of stations and the (variable) cost of operating the stations, that is (A-5) is true for GALBP (and, therefore, SALBP). Some of these issues, however, have recently been addressed: Graves and Lamar (1983), for instance, define a station selection and task assignment problem for automated assembly systems in which one of the optimization criteria is "cost", and Pinto, Dannenbring and Khumawala (1983) deal with the problem of choosing among processing alternatives, as well as the assignment of the tasks to the stations, while minimizing labor and fixed costs. The problems in which (A-5) is not true will be referred to as the *assembly line design problem* (ALDP). Thus, the main difference between GALBP and ALDP is that, in the latter, the choice problem includes technology/labor, based on fixed and variable costs (i.e., ALDP is a generalization of GALBP). The ALDP models and methods will not be reviewed here.

2.3. *SALBP and Other Related Combinatoric Optimization Problems*

In terms of practical problems, SALBP falls into the general class of *sequencing* and *scheduling problems* (Baker 1974). Sequencing problems are scheduling problems in which an ordering of the jobs/tasks completely determines the schedule. These scheduling problems are further complicated by the fact that the scheduling decisions are generally subject to 'precedence' constraints as well. The simplest example in this class is the *single resource scheduling problem,* with the resource being a machine/processor. This problem is closely related to ALBP: for instance, if the precedence constraints in SALBP-1 are replaced by the requirement that the immediate predecessors of each task must be assigned to an earlier station, the resulting problem becomes a *single resource constrained scheduling problem* (e.g., Garey, Graham, Johnson and Yao 1976). Another special case of SALBP-1 that has generated a lot of interest is known as the *bin packing problem* which simply is SALBP-1 without precedence constraints: a given collection of items (tasks) are to be packed into (assigned to) a minimum number of bins (stations) each with identical finite capacities (cycle time) (e.g., see Garey and Johnson 1981). Hence, SALBP-1 is a generalization of the bin packing problem (e.g., see Wee and Magazine 1982). A well-known problem closely related to the bin packing problem, and hence to SALBP-1, is the *knapsack problem:* there is now only one bin (knapsack) and only a subset of the given collection of items can be packed into that bin. Those items are to be chosen for packing while optimizing some criterion (e.g., Balas and Zemel 1980). And, finally, there is the well-known *partition problem:* partition a given collection of objects into a number of disjoint subsets while optimizing some prespecified criterion. The subsets of the task set designated for the respective stations is clearly a partition of the task set. Hence, SALBP is a reduction of the partition problem (Karp 1972).

Since the above problems are closely related to each other, it is possible to incorporate the solution methodology for one problem as a subroutine in the solution methodology for a related problem. For instance, Talbot and Patterson (1984) repeatedly solve knapsack problems to assist in the solution of SALBP-1 by eliminating inferior task assignments from consideration (§4.1.2) and Wee and Magazine (1981a) use bin packing heuristics to obtain lower bounds for solving SALBP (§4.1.3).

3. Mathematical Programming Formulations of SALB Problems

3.1. *Preliminaries and Notation*

According to both Tonge (1961) and Prenting and Thomopoulos (1974), Bryton (1954) was the first to give an analytical statement of ALBP. However, the first published analytical statement of the problem is due to Salveson (1955). We first present our notation:

Downloaded from informs.org by [141.161.91.14] on 05 September 2014, at 09:39 . For personal use only, all rights reserved.

RIGHTS LINK

I: task set ($I = \{1, 2, \ldots, i, \ldots, m\}$),
J: station set ($J = \{1, 2, \ldots, j, \ldots, n\}$),
$I(j)$: subset of tasks assigned to station j, $j \in J$,
t_i: process time of task i, $i \in I$,
$T_j = \sum_{i \in I(j)} t_i$ (i.e., work content of station j),
$W = \sum_{i=1}^{m} t_i$ (i.e., total task processing times),
T: cycle time,
$t_{\min} = \min_{i \in I} \{t_i\}$,
$t_{\max} = \max_{i \in I} \{t_i\}$,
$R = \{\text{all } (h, i) | h \text{ is an immediate predecessor of } i\}$ (i.e., the arc set of the precedence network),
$P(i) = \{h \in I | (h, i) \in R\}$ (i.e., the immediate predecessors of task i),
$P_a(i)$: {all predecessors of i},
$S(i) = \{h \in I | (i, h) \in R\}$ (i.e., the immediate successors of task i),
$S_a(i)$: {all successors of i},
$F = \{i \in I | S(i) = \varnothing\}$ (i.e., tasks with no successors),
n^*: the optimal (minimum) number of stations needed,
$n_{\min}$: a (known) lower bound on the optimal number of stations needed,
$n_{\max}$: a (known) upper bound on the optimal number of stations needed,
E_i: the smallest numbered station to which task i can be assigned (i.e., "early station" for task i) subject to precedence and cycle time restrictions,
L_i: the largest numbered station to which task i can be assigned (i.e., "late station" for task i) subject to precedence and cycle time restrictions,
I_f: a feasible subset of tasks with respect to T (i.e., $\sum_{i \in I_f} t_i \le T$),
S_f: a feasible sequence of tasks (with respect to precedence requirements),
$[x]^+$ = the smallest integer larger than or equal to x.

In his pioneering work, Salveson (1955) noted that *total slack* (TS) is a function of the number of stations n along the line: $\text{TS}(n) = \sum_{j=1}^{n} (T - T_j) = nT - W$. Since T and W are constants, TS is a linear function of n, and, therefore, it is minimized if the number of stations along the line is minimized. Also note that $[W/T]^+ \le n^* \le m$. These bounds are referred to as *theoretical minimum* and *maximum*, respectively, of the number of stations needed. Clearly, $n_{\min} \ge [W/T]^+$ and $n_{\max} \le m$.

3.2. *Alternative* 0-1 *Programming Formulations of SALBP*-1

Salveson (1955) formulated SALBP-1 as a linear programming (LP) problem encompassing all possible combinations of station assignments. His model, by definition, can result in split tasks and, therefore, may result in infeasible solutions. Bowman (1960) was first to provide a "nondivisibility" constraint, by changing the LP formulation to one of integer programming. (We should note that the dynamic programming method of Jackson 1956 did not allow split tasks either. See §4.3.) Bowman presented two different mathematical programs. In the first formulation, the decision variables represent the "number of time units devoted to a task at a station". In the second formulation, stations are not explicitly represented in the model: the decision variable represents the "clock time when a task started". Bowman then introduced a 0-1 variable to guarantee that the tasks do not use the same clock time (interference). We now present the first Bowman model, as modified by White (1961) who defined the following decision variable:

$$x_{ij} = \begin{cases} 1 & \text{if task } i \text{ is assigned to station } j, \\ 0 & \text{otherwise,} \end{cases} \qquad \forall\, i \in I \text{ and } \forall\, j \in J. \tag{1}$$

Note that, the first $n_{\min}$ stations need not be included in any "total cost" function to

Downloaded from informs.org by [141.161.91.14] on 05 September 2014, at 09:39 . For personal use only, all rights reserved.

be minimized. Furthermore, only tasks with no successors need positive coefficients in such an objective function because at least one of these tasks will be assigned to the last (largest numbered) station along the line. Let c_j be the "cost" of using station j such that;

$$c_{j+1} \geq Mc_j \qquad \forall\, j \in J - \{n_{\max}\} \tag{2}$$

where M is a sufficiently large positive integer (e.g., $M \geq m$). Note that this is *not an actual cost;* it is a form of penalty for using the station j. The purpose here is to make the latter stations exceedingly costly, specifically, one unit of later assignments is costlier than all the earlier assignment (i.e., "cost explosion"), as we see below.

The following 0-1 program, PROGRAM-BW, constructed by White (1961) and based on Bowman's formulation represents SALBP-1:

$$\text{minimize} \qquad z = \sum_{i \in F} \sum_{j=(n_{\min}+1)}^{n_{\max}} c_j x_{ij}, \tag{3}$$

$$\text{subject to:} \qquad \sum_{j=1}^{n} x_{ij} = 1 \qquad \forall\, i \in I, \tag{4}$$

$$\sum_{i=1}^{m} t_i x_{ij} \leq T \qquad \forall\, j \in J, \tag{5}$$

$$x_{ik} \leq \sum_{j=1}^{k} x_{hj} \qquad \forall\, k \in J \text{ and } \forall\, i \in I \text{ and } \forall\, h \in P(i), \tag{6}$$

$$x_{ij} = 0, 1 \qquad \forall\, i \in I \text{ and } \forall\, j \in J. \tag{7}$$

First of all, (7), of course, guarantees that each variable assume only values of 0 or 1, that is, a task cannot be split among two or more stations and is known as the *nondivisibility constraint.* The objective function (3) represents the "total cost" of using stations $n_{\min} + 1$, $n_{\min} + 2, \ldots, n_{\max}$. As noted by Bowman (1960, p. 387), "stations $(1, 2, \ldots, n_{\min})$ must certainly be used and need assume no costs. (Furthermore) only (tasks) with no succeeding (tasks) in an ordering need positive costs in the objective function, ...". (Bowman and White used $[W/T]^+$ and m, respectively as $n_{\min}$ and $n_{\max}$.) Relation (2) defines a very large premium on using an additional station when in fact all tasks can be accommodated without that station. Thus, the minimization of (6) will result in the minimum number of stations. Constraint (4), known as the *occurrence constraint,* guarantees that every task is assigned to a station and the *cycle time constraint* (5) guarantees that the work content of every station is at most the prespecified cycle time. Finally, (6), the *precedence constraints,* represent the technological sequencing requirements: if $x_{ik} = 1$, i.e., if task i is assigned to station k, then the RHS of (6) must assume a positive value (for each immediate predecessor h of i), i.e., each immediate predecessor h of i must be assigned to a station j such that $j \leq k$ so that task h is completed before the processing of task i starts. If, on the other hand, $x_{ik} = 0$, i.e., if task i is *not* assigned to station k, then the RHS of (6) need not be positive, i.e., the immediate predecessor h of i may or may not be assigned to the first k stations. Now consider some immediate predecessor h of i: if the RHS of (6) (for i and h) is zero, i.e., if task h has *not* been assigned to any one of the first k stations, then $x_{ik} = 0$ (i.e., the LHS of (6) is equal to zero), that is, task i cannot be assigned to any one of these k stations either. On the other hand, if the RHS of (6) is positive, i.e., if, say $x_{ik'} = 1$, $(k' \leq k)$, then task i could be assigned to station k', $k' + 1, \ldots, k$. Note that $x_{ik'} = 1$ implies that the RHS of (6) for $k = 1, 2, \ldots, (k' - 1)$ and i and h is zero and, therefore, task i cannot be assigned to the stations $1, 2, \ldots, (k' - 1)$. The original formulation of Bowman had another set of constraints to avoid splitting of the tasks between the stations. However, White showed that those constraints were redundant.

Clearly, the coefficients in (3) grow rapidly and pose numerical stability problems due to the magnitude of the difference between the smallest and the largest one (for instance, if there are $m = 20$ tasks and if $n_{\max} - n_{\min} = 10$). Partly because of this

Downloaded from informs.org by [141.161.91.14] on 05 September 2014, at 09:39 . For personal use only, all rights reserved.

Downloaded from informs.org by [141.161.91.14] on 05 September 2014, at 09:39 . For personal use only, all rights reserved.

reason other objective functions have been proposed over time. For instance, Thangavelu and Shetty (1971) use the following objective function coefficients:

$$c_{ij} = \begin{cases} t_i[\sum_{h\in F} t_h + 1]^{(j-n_{\min}-1)} & \text{for } j = (n_{\min}+1), \ldots, n_{\max} \text{ and } i \in F, \\ 0 & \text{otherwise,} \end{cases} \tag{8}$$

where $n_{\min} = [W/T]^+$. Thus, (3) can be replaced by

$$z = \sum_{i=1}^{m} \sum_{j=1}^{n_{\max}} c_{ij}x_{ij} \tag{9}$$

where c_{ij} is as in (8). We shall refer to the Thangavelu and Shetty (1971) formulation as PROGRAM-TS. An alternative objective function was proposed by Patterson and Albracht (1975) as well. They define, for each task i, the *earliest station* E_i and the *latest station* L_i that task i can be assigned to, based upon the precedence relations:

$$E_i = \begin{cases} 1 & \text{for } (t_i + \sum_{h\in P_a(i)} t_h) = 0, \\ [(t_i + \sum_{h\in P_a(i)} t_h)/T]^+ & \text{otherwise,} \quad i = 1, 2, \ldots, m. \end{cases} \tag{10}$$

$$L_i = \begin{cases} n & \text{for } (t_i + \sum_{h\in S_a(i)} t_h) = 0, \\ n + 1 - [(t_i + \sum_{h\in S_a(i)} t_h)/T]^+ & \text{otherwise,} \quad i = 1, 2, \ldots, m. \end{cases} \tag{11}$$

Thus, in the Patterson and Albracht (1975) formulation of SALBP-1, which we shall refer to as PROGRAM-PA, instead of (3), the objective function is as follows:

$$z = \sum_{j=E_d}^{n_{\max}} (n_{\max} + 1 - j)x_{dj} \tag{12}$$

where task d is a (unique) dummy "last" task with $t_d = 0$ and $P(d) = F$ (i.e., a dummy "last" task is introduced, together with dummy arcs connecting d to each task $i \in F$, so that each original "last" task is now an immediate predecessor of the newly introduced dummy "last" task).

Improvements on the Bowman-White formulation are not limited to the objective function. First of all, Thangavelu and Shetty (1971) noted that the precedence constraints (6) can be replaced more generally with the following:

$$\sum_{j=1}^{n_{\max}} (r_{ij}x_{ij} - r_{hj}x_{hj}) \geq 0 \tag{13}$$

where $r_{ij} \geq r_{hj} \geq r_{i,j+1}$ $(j = 1, \ldots, n_{\max})$ with $r_{i,n_{\max}+1} \equiv 0$ because, any solution satisfying the occurrence constraints (7) is of the form

$$x_{ij} = \begin{cases} 1 & \text{for some } j = r, \\ 0 & \text{otherwise,} \end{cases} \quad \text{and} \quad x_{hj} = \begin{cases} 1 & \text{for some } j = k, \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

Hence precedence constraints are satisfied if and only if $r \leq k$ (i.e., task i must be assigned to station k, or a preceding station, if task h is assigned to station k). Thus,

$$\sum_{j=1}^{n_{\max}} (n_{\max} - j + 1)(x_{ij} - x_{hj}) \geq 0 \qquad \forall\, (i, h) \in R. \tag{15}$$

For otherwise, i.e., if $k < r$, we would have $(n_{\max} - k + 1)(x_{ik} - x_{hk}) + (n_{\max} - r + 1) \times (x_{ir} - x_{hr}) = (n_{\max} - k + 1)(0 - 1) + (n_{\max} - r + 1)(1 - 0) = (k - r)$. But $(k - r) < 0$ by definition, thereby violating (15). If, on the other hand, $k \geq r$, then (15) obviously holds. It can also be shown easily that (15) is violated only if $k < r$.

Patterson and Albracht (1975) also propose an alternate relation for the precedence requirements:

$$\sum_{j=E_i}^{L_i} j(x_{ij}) \leq \sum_{k=E_h}^{L_h} k(x_{hk}) \tag{16}$$

where i is an immediate predecessor of h and $L_i \geq E_h$. Now suppose task h is assigned to station a and task i is assigned to station b, such that $a < b$ (i.e., suppose $x_{ib} = x_{ha}$). Then, in (16), the LHS is bx_{ib} and the RHS is ax_{ha} or, equivalently, b and a, respectively. But, by definition, $b > a$ and, therefore, (16) is violated. On the other hand, if $a \geq b$, then (16) clearly is satisfied.

Thangavelu and Shetty (1971) next tackled the occurrence constraints. Rather than replacing the m equations (4) (the occurrence constraints) with $2m$ inequalities, Thangavelu and Shetty replaced (4) with the following inequalities:

$$1 - \sum_{j=1}^{n_{\max}} x_{ij} \geq 0 \qquad \forall\, i \in I \qquad \text{and} \qquad -m + \sum_{i=1}^{i=m} \sum_{j=1}^{n_{\max}} x_{ij} \geq 0. \tag{17}$$

The first inequality in (17) guarantees that each task is assigned at most once and the second one guarantees that at least m assignments are made. Thus (17) is equivalent to (4).

Similarly, Patterson and Albracht (1975) also note that the occurrence constraint (4) using "early" and "late" stations (see (10) and (11)) is satisfied if

$$\sum_{j=E_i}^{L_i-1} x_{ij} \leq 1 \qquad \forall\, i \in I \tag{18}$$

is satisfied and

$$x_{iL_i} = 1 - \Big(\sum_{j=E_i}^{L_i-1} x_{ij}\Big) \qquad \forall\, i \in I. \tag{19}$$

Relation (18) is clearly valid: task i need not necessarily be assigned to its late station and (19) allows for the possibility that i is assigned to its late station (when the LHS of (18) is zero, then the RHS of (19) is equal to 1).

These manipulations significantly reduce the size of the Bowman and White formulations, as we see in Table 1. An alternative formulation would be through the introduction of an additional 0-1 variable:

$$\delta_j = \begin{cases} 1 & \text{if station } j \text{ is used,} \\ 0 & \text{otherwise.} \end{cases} \tag{20}$$

Then, the objective function would be as follows:

$$\text{minimize} \qquad z = \sum_{j=1}^{n_{\max}} \delta_j. \tag{21}$$

But then two additional constraints would be needed. First of all,

$$\sum_{i=1}^{m} t_{ij} x_{ij} \leq T\delta_j \qquad \forall\, j \in J \tag{22}$$

must hold so that the decision variable x_{ij} can assume a positive value only when the corresponding station has been opened (i.e., when $\delta_j = 1$). Secondly, a new station should not be opened unless the preceding station has been opened:

$$\delta_{j+1} \leq \delta_j \qquad \forall\, j = 1, 2, \ldots, n_{\max} - 1. \tag{23}$$

Granted, the number of variables is increased but, in this formulation, the objective

TABLE 1

Dimensions of Three Different 0-1 *Programming Formulations*
(*Obtained from Patterson and Albracht* 1975)

	PROGRAM-BW	PROGRAM-TS	PROGRAM-PA
Relations	(3)–(7)	(5),(7),(9),(15),(17)	(5),(7),(12),(16),(18),(19)
Variables	56	56	29
Constraints	71	24	23

Downloaded from informs.org by [141.161.91.14] on 05 September 2014, at 09:39 . For personal use only, all rights reserved.

Downloaded from informs.org by [141.161.91.14] on 05 September 2014, at 09:39 . For personal use only, all rights reserved.

function coefficients are not irregular. We have successfully used a similar objective function for minimizing the number of broadcast frequencies to meet demand while avoiding electromagnetic interference (Baybars 1982).

3.3. *A General Integer Programming Formulation of SALBP*-1

In a recent paper Talbot and Patterson (1984) propose a new formulation, one not involving binary decision variables. The decision variable in this formulation is defined to be the "number of the station that task i is assigned to", denoted by x_i $\forall\ i \in I$. PROGRAM-TP below is the Talbot and Patterson formulation:

$$\text{minimize} \qquad z = x_m \tag{24}$$

$$\text{subject to:} \qquad \textstyle\sum_{i \in I(j)} x_{ij} \le T \qquad \forall\ j \in J, \tag{25}$$

$$x_h \le x_i \qquad \forall\ h \in P(i) \text{ and } \forall\ i \in I, \tag{26}$$

$$x_i \ge 0 \text{ and integer} \qquad \forall\ i \in I, \tag{27}$$

where m is the unique terminal task, dummy if necessary. Clearly, the main advantage of this formulation is that the number of variables equals the number of tasks. Furthermore, the objective function has only one term, namely the station number of the last (dummy) task in the precedence network. Other formulations of SALBP-1 have been proposed by Klein (1963) and Gutjahr and Nemhauser (1964). These will be discussed in §4.2.

3.4. *A* 0-1 *Programming Formulation of SALBP*-2

When the number of the stations n along the line is fixed and the goal is to minimize the cycle time T, the problem then becomes a minimax problem. That is, the largest work content must be minimized. PROGRAM-MT below represents SALBP-2:

$$\text{minimize} \qquad T \tag{28}$$

$$\text{subject to:} \qquad \textstyle\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall\ i \in I, \tag{29}$$

$$T \ge \textstyle\sum_{i=1}^{m} t_i x_{ij} \qquad \forall\ j \in J, \tag{30}$$

$$x_{ik} \le \textstyle\sum_{j=1}^{k} x_{hj} \qquad \forall\ i \in I,\ \forall\ k \in J \text{ and } \forall\ h \in P(i), \tag{31}$$

$$x_{ij} = 0, 1 \qquad \forall\ i \in I \text{ and } \forall\ j \in J. \tag{32}$$

Note that (29), (31) and (32) are the *occurrence, precedence* and *nondivisibility* constraints, respectively (as are (4), (6) and (7), respectively, in PROGRAM-BW). The objective function (30) represents the cycle time and is to be minimized. Since n, the number of stations along the line, is fixed, the cycle time T is bounded from below by the work content of each station j (the RHS of (30)). Also note that (29) and (32) guarantee that T will be positive.

3.5. *On the Complexity of SALB Problems and Measures for Complexity*

First of all it is well known that both SALBP-1 and SALBP-2 are $\mathcal{NP}$-hard because the partition problem is known to be $\mathcal{NP}$-hard (Karp 1972) and, as we noted earlier, SALBP is a reduction of the partition problem. Clearly, there are $m!$ possible sequences of tasks in SALBP. If there exist r precedence requirements, that is if the precedence network has r arcs, then there are roughly $m!/2^r$ distinct feasible sequences (a sequence of tasks that can be processed in the indicated order without prior completion of any other task is called a *feasible sequence*). Thus, the magnitude of the feasible sequences/

RIGHTSLINK

subsets is an important factor limiting the performance of the exact solution methodologies.

Over the years a number of measures have been proposed for the complexity of the line balancing problems. For instance, for a given SALB problem P, Mastor (1970) proposes the measure called *order strength,* which we denote by $\alpha(P)$, namely the number of precedence relations (i.e., $|R|$) divided by the total possible number of precedence relations in P. A similar measure is what Dar-El (1973) calls the *flexibility ratio,* which we denote by $\beta(P)$, namely, the number of zero entries in the precedence matrix of P divided by the number of entries in that matrix. Clearly, both invariants $\alpha(P)$ and $\beta(P)$ essentially measure the magnitude of the feasible sequences/subsets in P: $\alpha(P) = 0$ implies that there exist no precedence relations in P and $\alpha(P) = 1$ implies a unique sequence. *In general* the CPU time for solving a SALB problem increases as its order strength decreases, and, therefore, $\alpha(P)$ is a good predictor of CPU times (see, for instance, Mastor 1970 and Johnson 1981).

Yet another measure, the *west-ratio,* proposed by Dar-El (1973), is in terms of the nature of the optimal solution. For a given SALB problem P, the west ratio, denoted by $\gamma(P)$, is the number of tasks per station in an optimal solution and, *in general,* the CPU time for solving P is an increasing function of γ. However, west ratio is not a strong indicator by itself (Wee and Magazine 1981a note, for instance that, the *normalized interval,* namely $(t_{\min}/T, t_{\max}/T)$ was an important factor in the implementation of their method).

4. Exact Algorithms for Solving SALBP-1

4.1. *Integer Programming* (*IP*) *Algorithms*

Among the four primary approaches to solving IP problems (cutting plane techniques, enumerative techniques, partitioning methods and group theory), only the enumerative techniques (i.e., "branch and bound" (B&B) or "search" methods) have been extensively applied to SALBP-1. A B&B procedure is, typically, either a *newest node search method* or a *frontier search method.* The former creates arcs from the set of the most recently created nodes and it is also known as "depth-first search" or "backtracking" method. In the frontier search methods, the arcs are created from the set of all exposed nodes and it is also known as "most promising node" or "jumptracking" method. Some versions of the frontier search methods are also known as "breadth-first search methods".

4.1.1. *A General IP Algorithm Applied to SALBP-*1. In this section we discuss a general IP algorithm applied to SALBP-1 by Thangavelu and Shetty (1971) who solved their 0-1 program, namely PROGRAM-TS, by applying the additive algorithm of Balas (1965), as presented by Geoffrion (1967). In this newest node search method there are two subroutines, one for augmenting the partial solution if it may lead to a feasible completion better than the incumbent feasible solution, and the other one for backtracking and record-keeping whenever a feasible completion better than the incumbent is obtained or when it can be shown that such a solution does not exist. Thangavelu and Shetty add a conditional feasibility test to the Geoffrion algorithm (a constraint is *conditionally feasible* if its feasibility is contingent upon certain of the variables taking on particular binary values). The test permits ready augmentation of the partial solution retaining feasibility so that the implicit enumeration process is expedited. They start with a feasible solution, obtained by the heuristic procedure of Helgeson and Birnie (1961), from which they determine the optimal solution.

4.1.2. *Specialized Algorithms Based on General IP Methods.* The above method is a general IP method and, therefore, relies on IP theory and codes. We next discuss

Downloaded from informs.org by [141.161.91.14] on 05 September 2014, at 09:39 . For personal use only, all rights reserved.

Downloaded from informs.org by [141.161.91.14] on 05 September 2014, at 09:39 . For personal use only, all rights reserved.

two specialized methods which also rely on IP techniques, but are specialized for SALBP-1: Patterson and Albracht (1975) have proposed a search method for examining sequences of 0-1 programs for feasible solutions. As discussed earlier, in their formulation they use the concept of early and late stations for each task, thereby reducing the number of variables. Furthermore, they eliminate the occurrence constraints, use conditional feasibility tests for the precedence constraints, and use a binary infeasibility test for the cycle time constraints. This method is based on general IP codes and theory: they strengthen several steps of the Geoffrion (1967) enumeration process for fathoming a partial solution and the search is shortened by exploiting the special structure of the problem. In the search algorithm, they use the lower bound $[W/T]^+$ on n^*. They vary the number of stations n according to $n_k = [W/T]^+ - 1 + k$, $k = 1, 2, \cdots$ until a feasible solution is found. Alternatively, they vary the station number n according to $n_k = [W/T]^+ - 1 + F_k$, $k = 1, 2, \ldots$, where F_k is the Fibonacci number until a feasible solution is found. The interval is searched using a binary search procedure to determine the optimal number of stations.

Talbot and Patterson (1984) have recently constructed an IP algorithm to solve PROGRAM-TP which systematically evaluates all possible task assignments to the stations and, like the Thangavelu and Shetty (1971) method, is based on the implicit enumeration algorithm of Balas (1965). As noted earlier, Talbot and Patterson use integer variables rather than binary variables and this results in significant reduction in computer storage requirements (i.e., the number of variables equals the number of tasks). Significant portions of the enumeration process are performed implicitly by using tests; constraints are not explicitly formulated in the procedure; precedence relations are maintained by directly testing an immediate predecessors array and the cycle time restrictions are satisfied through the evaluation of an idle time vector. Similar to those given by Patterson and Albracht (1975), Talbot and Patterson also define "early" and "late" stations for each task:

$$E_i = [(t_i + \textstyle\sum_{h\in P_a(i)} t_h)/T]^+ \quad \text{for } i = 1, 2, \ldots, m, \tag{33}$$

$$L_i = n_{\max} - [(t_i + \textstyle\sum_{h\in S_a(i)} t_h)/T]^+ \quad \text{for } i = 1, 2, \ldots, m-1 \quad \text{and} \quad L_m = n_{\max} - 1, \tag{34}$$

In (34) above, the upper bound on the terminal task m is one station less than the current best known solution $n_{\max}$. In assigning a task to a station, these bounds are used directly to specify the range of the integer values that the variables x_i can assume. (In Patterson and Albracht 1975, the early/late station bounds (10) and (11) are used to reduce the number of 0-1 variables in the problem formulation.) Rather than using binary infeasibility tests, the order in which the variables are considered for augmentation is prespecified by the node numbering scheme. Talbot and Patterson exploit the problem structure to expedite the fathoming and branching process. Backtracking is done in two ways: using "chains" and/or "network cuts". A *chain* is an ordered set of consecutively numbered tasks that are either immediate predecessors or successors of adjacent members of the set. When such chains exist, a considerable amount of searching is avoided by directly backtracking to the highest numbered task not in the chain. They also use an artifice called a *network cut:* a cut is (a) a station number that is used to identify when certain fathoming and backtracking rules may be applied and (b) a parameter used in the application of the rules. A station s is a *cut* if there exists a task i with $L_i = s + 1$ and there exists no task $h > i$ such that $L_h \leq s$. Talbot and Patterson discuss two fathoming rules and two additional expediting rules based on a knowledge of the minimum amount of idle time which must be present in each station, determined by solving an imbedded knapsack problem.

RIGHTSLINK

4.1.3. *Specialized Branch and Bound Algorithms.* The feasible solutions to SALBP-1 can be represented by a tree in which each path corresponds to a feasible solution, with each arc representing a station. The first algorithm for SALBP-1 was constructed using this notion of a tree by Jackson (1956) (see §4.3). Following Jackson, a station is termed *maximal* if no task can be assigned to it without violating the precedence and cycle time constraints. Jackson showed that an optimal solution exists in a tree whose arcs represent only maximal stations. Using a Jackson (1956) type of an enumeration tree, Van Assche and Herroelen (1979) constructed a frontier search method but one that is not based on IP theory or codes. They present a tree search procedure, dominance rules, bounding arguments and branching heuristics with a node representing the assignment of tasks to a single station. The procedure starts with an empty station and moves down the unordered list. The problem associated with each node is one of assigning the remaining tasks to the remaining stations. One checks if there is an immediate solution or not. If none exists, one branches into a number of descendant nodes corresponding to the feasible undominated next station assignments and computes a lower bound, namely the lower bound on the remaining number of stations to be generated in order to obtain a solution for each such node. Let $n(k)$ be the number of stations used at level k of the procedure and let $T_{n(k)}$ be the sum of the process times of the tasks assigned to those $n(k)$ stations, i.e., let $T_{n(k)} = \sum_{j=1}^{n(k)} T_j$. The node "lower bound" for each node of the search tree is $n(k) + [(W + T_{n(k)})/T)]^+$. For the initial node they use $[W/T]^+$ as the lower bound. Then the node with the smallest lower bound is chosen for the next iteration. They also define a "penalty" associated with each node, namely the average time that must be assigned to the remaining stations if one is to obtain a solution, namely $(W - T_{n(k)})/[(W - T_{n(k)})/T]$.

The method also uses an extensive set of tie-breaking heuristic rules to differentiate between the nodes with the same lower bound and penalty. If an immediate solution is found for any node in the tree, then a solution is generated for all its ancestors. The assigned tasks are combined to form a new node in the B&B tree if and only if the tasks of that station are not a proper subset of the tasks of some other station generated during that iteration and it has not been eliminated as a result of dominance tests. A node corresponding to a sequence of station assignments along the path of the search tree leading to the particular node is eliminated if it is a subset of a sequence corresponding to another node (still not eliminated) at the same level of the search tree (the proof of this rule is in Jackson 1956).

Wee and Magazine (1981a) also report a B&B method and, again, one not based on general IP codes or theory. They use two simple heuristics termed IUFFD and IUBRPW to generate $n_{\max}$. The first one is a variation of the well-known "bin packing" heuristic *first-fit-decreasing* (FFD) rule (see Garey and Johnson 1981 and Wee and Magazine 1982). In the FFD rule method the tasks are listed in nonincreasing order of process times (i.e., similar to the "largest candidate rule" heuristic method of Moodie and Young 1965 for SALBP-1). A packing is then built by treating each task in succession and assign it to the lowest indexed station with available time. This is one of the most efficient heuristics for the bin packing problem (Wee and Magazine 1981a). Wee and Magazine use a variation of this rule: after the assignment of a task, they *immediately update* the set of available tasks and arrange them in nonincreasing order of the process times (hence the name IUFFD). The second heuristic is a backward recursive positional weight and again immediately updated (hence the name IUBRPW). This heuristic differs from IUFFD as follows: instead of using the task process time of a task i, they use BRPW(i), the sum of the process times of all chains with task i as the head (thus, this is essentially a reverse application of the "rank positional weight"

Downloaded from informs.org by [141.161.91.14] on 05 September 2014, at 09:39 . For personal use only, all rights reserved.

method of Helgeson and Birnie 1961 for SALBP-1). The solutions obtained in this fashion result in good bounds on the optimal number of stations needed.

Wee and Magazine use an enumeration tree similar to that of Jackson (1956) and Van Assche and Herroelen (1979): each node of the tree is a maximal feasible assignment of tasks to a station. At level k the upper bound for a node is the number of stations used (i.e., $n(k)$) plus number of stations required for the unassigned tasks determined by a heuristic method. The lower bound, on the other hand, is the number of stations used plus the theoretical minimum for the unassigned tasks. They start with a trial number of stations and if they can find a solution using that many stations, they reduce the number of stations by one and repeat the procedure. In their method, the starting node, node 0, is a null set and the descendant nodes of node 0 are all the possible maximal feasible assignments to the first station. To branch from a node with depth i, they generate all maximal feasible assignments to station $(i + 1)$. The rationale for their use of the concept of maximal feasible assignments for branching is as follows: let y and z be two descendant nodes of some node x and let F_y and F_z be the two corresponding feasible but not necessarily maximal assignments such that F_y is a subset of F_z. Then the optimal number of stations required based on y is greater than or equal to that based on z. Thus, there is no need to branch to y. (This is in fact an argument of Jackson 1956 and also used by Van Assche and Herroelen 1979 and, as we shall see, by Johnson 1981 as well.) They select the (unfathomed) branch with the smallest accumulated idle time as the new branching node. Various dominance tests are also given.

Johnson (1973) constructed a newest-node search B&B algorithm in which an arc represents a station. This algorithm does not rely on general IP theory or codes, but rather uses the "maximal station assignment" algorithm of Jackson (1956). Johnson (1981) developed an improved version of his 1973 algorithm differing only in the bounding mechanism. In Johnson (1981) new arcs are created from the set of the most recently created nodes: beginning with the first station, Johnson finds a feasible solution by choosing each arc to minimize idle time at that station and works through the last station of a particular solution. When a feasible solution is reached, the solution tree consists of one complete path and a number of end nodes adjacent to the nodes on that path. This solution is recorded as the incumbent best solution. Then, one backtracks to the node of the most recently created arc α not on the path. The path following α is deleted and the most recently created arc is used as the new path. Johnson uses the following bounds on n^*: $B_1 = [W/T]^+$. Let $l = [q/B_1]^+$. For $k = 2, \ldots, l$, Johnson then defines $B_k = [(\#\text{ of tasks with } t_i > (T/k))/(k - 1)]^+$. Then the bound is chosen as follows: $n_{\min} = \max_{k=1,2,\ldots,l} \{B_k\}$. The method is terminated when the whole tree has been implicitly or explicitly considered.

4.2. *Methods Based on Other IP Problems and Techniques*

An alternate approach to solving SALBP-1 was proposed by Klein (1963) who presented two different formulations of the problem. The first formulation requires the solution of a series of *assignment problems*. Klein proposes to generate all feasible sequences of tasks for the given precedence diagram. Then, the assignment minimizing the total idle time can be determined by solving the associated assignment problem. This has to be repeated for each feasible sequence and the optimal solution to SALBP would be among the solutions to the series of the assignment problems. His second formulation is very much like the next method that we discuss, due to Gutjahr and Nemhauser (1964). Following Held, Karp and Sharesian (1963), a *feasible subset* is a subset of the tasks that can be processed without prior completion of any other task

Downloaded from informs.org by [141.161.91.14] on 05 September 2014, at 09:39 . For personal use only, all rights reserved.

Downloaded from informs.org by [141.161.91.14] on 05 September 2014, at 09:39 . For personal use only, all rights reserved.

and in any order that satisfies the precedence relations and a *feasible (sub)sequence* is a special case and was defined as a subsequence of the tasks that can be processed in the indicated order without prior completion of any other task. Thus, the latter is necessarily an *ordered* set, unlike the former. Gutjahr and Nemhauser use this concept of "feasible subsets". The method is a *shortest path* based network technique. Gutjahr and Nemhauser begin by generating a directed network based on the precedence diagram the nodes of which correspond to feasible subsets, with the source node being the null set and the sink node being the set of all tasks. The "time" τ_i associated with node i is simply the sum of the process times of the tasks represented by that node. Let $S(i)$ denote the subset of tasks represented by node i. An arc (i, j) is in the directed network if and only if $S(i)$ is a subset of $S(j)$ and $\tau(j) - \tau(i) \leq T$. Thus, an arc (i, j) corresponds to a station assignment of all tasks in $S(j)$ but not in $S(i)$. As a result of this, there is a one-to-one correspondence between source-sink paths and feasible assignments. Thus, finding the shortest path in this directed network is equivalent to finding the minimum number of stations required.

Another method is due to Charlton and Death (1969) who defined a general B&B method for machine scheduling and applied it to PROGRAM-BW. In their frontier search method each arc represents the assignment of a single task. The sequence with the smallest lower bound is chosen to be examined next and new arcs are created from the set of all exposed nodes. They simply enumerate all possible sequences of tasks in such a way that each sequence is considered only once. At each stage, a set of partial sequences of tasks at each station is extended by adding a previously unassigned task to the sequence of one of the stations, i.e,. a new task is assigned. Each iteration of the algorithm requires the solution of a *critical path problem* which provides the "early" and "late" stations for the tasks, thereby yielding a lower bound. The solution to the critical path problem provides the starting times of all the tasks assigned at that stage, as well as the earliest start times for all the unassigned tasks. The lower bound used is determined as follows: for each station the idle time is found. To that they add the process times for all the tasks and they find the minimum number of stations whose total availability exceeds this sum. The lower bound is then the maximum of this value and the number of stations already in use at that stage.

4.3. *Dynamic Programming* (*DP*) *Methods*

The very first algorithm for solving SALBP-1, a DP method, was constructed by Jackson (1956) (but his procedure was not formulated using the conventional DP terminology; such a presentation is provided by Held, Karp and Sharesian 1963). Jackson starts by generating all feasible assignments to the first station (one of these feasible assignments will obviously be part of the optimal solution). Then one generates all feasible assignments to the second station, given the first station assignments. Then, for each first-second station combination, all feasible solutions are constructed for the third station, etc. The process is then repeated, each time adding another station. The method is terminated with an optimal solution when a value of n^* is reached for which all tasks can be assigned to n^* stations. Jackson uses dominance arguments to eliminate inferior feasible assignments: after generating all feasible assignments, these sets are compared to eliminate subsets that contain exactly the same tasks as other subsets and to eliminate the dominated sets (i.e., a *dominated set* is one that contains fewer tasks than another subset and does not contain any tasks that are not also in the other subset). Jackson's method does not require the generation of all feasible sequences but still uses exhaustive search of all remaining possibilities. (Jackson 1956 actually presented two methods: in his modified method, he uses dominance arguments which are applied to sets which are not proper subsets of another set.)

A few years later a new DP algorithm was reported by Held and Karp (1962) and described in detail in Held, Karp and Sharesian (1963). As noted previously (§4.2), Held, Karp and Sharesian introduced the notion of "feasible subsets" and "feasible sequences". In their DP method, associated with each feasible sequence σ is a particular assignment of tasks to stations called the "induced assignment of σ". This assignment minimizes the number of stations required to process the tasks in σ in the indicated order and can be obtained simply by filling stations until all tasks are assigned. Associated with each subsequence they defined as the "cost of a feasible subsequence" the number of stations that the subsequence requires plus the time required in the last station. Thus the cost of a feasible subset is the minimum of the costs of its associated feasible subsequences. The cost of each feasible subset is computed recursively from subsets of 1, 2, . . . , $m - 1$ tasks in the partial ordering (precedence relations). They use a recursive relationship to get the costs of the subsets with 2 tasks from those with one task and, in general, s tasks from those with $s - 1$ tasks, until the cost of the entire task is obtained. This is the minimum cost for the line since the task set has all the feasible sequences associated with it. The feasible sequence to be used for the assignment of tasks to the stations is also determined recursively; after each task is deleted from the partially ordered set, the cost of the remaining feasible subset must equal the cost of one of the feasible subsets determined using the recursive cost relations. The necessary and sufficient condition for the optimality of the induced assignment of a feasible sequence, then, is the cost of a feasible sequence must equal the cost of the feasible sequence without the last task plus the costs that are incurred by adding the last task to the set.

More than a decade elapsed before other significant DP approaches were reported. This lag was probably due to the severe computational and storage requirements of the DP methods. Schrage and Baker (1978) proposed an efficient method for generating all feasible sets and a method for assigning to each feasible set a label that can be used as a physical address for storing information about the set within the framework of DP approach to sequencing problems with precedence constraints, a special case of which is SALBP. The feasible sets are generated using a lexicographic ordering scheme. For the task labeling scheme, let $L(S)$ and $L(i)$ be the labels assigned to a feasible set S and a task i, respectively. The tasks are labeled, using a sequence $i_1, i_2, \ldots, i_n$, so as to ensure that $L(S)$ will uniquely identify S. The task labels are then used to obtain the set labels ($L(S) = \sum_{i\in S} L(i)$). Let $\mathcal{L} = L(\{1, 2, \ldots, n\})$. Since $\mathcal{L}$ dictates the storage requirements, Kao and Queyranne (1982) propose a variant of the Schrage and Baker (1978) procedure, and another approach related to the Schrage and Baker method, for chain decomposition of the precedence relations. Kao and Queyranne propose these approaches for their version of the DP formulation of SALBP. For a feasible set S, they define the minimum cost function $T(S) = (r - 1)T + f_r$ where r is the minimum number of stations needed for all the tasks in S while not violating the precedence relations and f_r is the sum of the process times of the tasks assigned to the last station under an optimal grouping of all tasks in S. Let $Q(S)$ denote the set of tasks in S with no successors in S. The Kao and Queyranne DP recursion, then, is as follows: $T(S) = \min_{i\in Q(S)} [T(S), T(S - \{i\}) + \Delta(T(S - \{i\}), t_i)]$ with $T(\varnothing) = 0$ and where $\Delta = y$ if $y \leq T - (x - T[x/T])$ and $T - (x - T[x/T]) + y$, otherwise. They initially set $T(S) = \infty$ for all $S \neq \varnothing$. This formulation is identical to that of Held, Karp and Sharesian (1963) when the $T(S)$ term is dropped from the recursive relation. The recursive computations can be carried out by using either a "pulling" or a "reaching" approach (Denardo and Fox 1979). Kao and Queyranne (1982) discuss two "pulling" approaches, namely the Schrage and Baker (1978) method and their own storage-saving variant as well as the "reaching" based procedure of Lawler (1979). The authors' empirical results suggest that the Lawler approach is computationally more efficient both in terms of processing times and storage requirements. Kao and

Downloaded from informs.org by [141.161.91.14] on 05 September 2014, at 09:39 . For personal use only, all rights reserved.

Queyranne (1983) have also presented recently hybrid DP and B&B models, suggesting that such an approach may be more efficient relative to individual DP and B&B approaches.

4.4. *A Summary of the Reported Computational Experience*

A considerable amount of computational experience has been reported by Van Assche and Herroelen (1979), Johnson (1981), Wee and Magazine (1981a) and Talbot and Patterson (1984) but none by Salveson (1955), Jackson (1956), Bowman (1960), White (1961), Jaeschke (1962), Klein (1963), Gutjahr and Nemhauser (1964), Mertens (1967) and Charlton and Death (1969). In Table 2, we present a summary of the computational experience reported on a number of well-known problems.[1]

Since the experimentations have been performed on different types of problems, for different cycle times, and using different computers,[2] a *meaningful comparison of the*

TABLE 2

Computation Times of Select Algorithms for Select Problems[3]

Tasks	Problem Source	Methodology	Cycle Times	Mean CPU Sec
11	Jackson (1956)	Thangavelu and Shetty (1971)	10, 12	0.8284
		Patterson and Albracht (1975)	10, 12	0.0734
		Schrage and Baker (1978)	10, 14,21	0.1876
		Van Assche and Herroelen (1979)	10, 12	0.0085
		Wee and Magazine (1981a)	10, 14, 21	0.0407
		Talbot and Patterson (1984)	7, 9, 10, 13, 14, 21	0.3176
21	Mitchell (1957)	Thangavelu and Shetty (1971)	14, 18, 20, 21	0.1025
		Patterson and Albracht (1975)	14, 18, 19, 20, 21	0.1036
		Schrage and Baker (1978)	14, 26, 39	1.3497
		Van Assche and Herroelen (1979)	14, 18, 19, 20, 21	0.0186
		Wee and Magazine (1981a)	26, 39	0.0538
		Talbot and Patterson (1984)	14, 15, 21, 26, 35, 39	0.3180
45	Kilbridge & Wester (1962)	Thangavelu and Shetty (1971)	69	4.1193
		Patterson and Albracht (1975)	69, 92, 138, 184	1.0692
		Schrage and Baker (1978)	97, 110, 164	175.0+
		Van Assche and Herroelen (1979)	69	1.7570
		Wee and Magazine (1981a)	110, 164	0.1210
		Talbot and Patterson (1984)	57, 79, 92, 110, 138, 184	8.3152
70	Tonge (1961)	Patterson and Albracht (1975)	358	27.1543
		Schrage and Baker (1978)	173, 176, 346, 468	175.0+
		Van Assche and Herroelen (1979)	83, 86, 89, 95, 176, 179, 182, 346, 349, 352, 355, 358	2.6064
		Wee and Magazine (1981a)	346, 468	0.1695
		Talbot and Patterson (1984)	364, 410, 468, 527	7.6708

[1] Neither Held, Karp and Sharesian (1963) nor Johnson (1981) report experimentation with this problem set.

[2] In terms of mainframe KOPS ratings, IBM 370/158-1 is approximately 1.09 faster than UNIVAC 1108. On the other hand, IBM 370/168-1 and AMDAHL 470/V8 are approximately 2.77 and 7.69 times faster, respectively, than IBM 370/158-1 (see Lias 1980).

[3] Notes: (a) The CPU times are averages for the given cycle times and adjusted to IBM 370/158-1 CPU times (Patterson and Albracht 1975 performed their experimentations using IBM 370/168-1; Talbot and Patterson 1984 using AMDAHL 470/V8; and Thangavelu and Shetty 1971 using UNIVAC 1108. Other methods were implemented on IBM 370/158-1), (b) Schrage and Baker (1978) times reported here are based on the implementation of the method by Wee and Magazine (1981a) (all other times are reported by those who constructed the algorithms), (c) The Talbot and Patterson (1984) times are for runs "with network cuts", and (d) the CPU times reported by Talbot and Patterson and Wee and Magazine *include* the input/output times as well, whereas others do not.

Downloaded from informs.org by [141.161.91.14] on 05 September 2014, at 09:39 . For personal use only, all rights reserved.

methods is difficult. We have, therefore, adjusted the reported CPU times to that of an IBM 370/158-1. Furthermore, because the studies do not cover all the cycle times, comparison is not straightforward. For instance, the Talbot and Patterson (1984) method is seemingly more efficient if one solely pays attention to the times given in Table 2. However, for the 70-task problem of Tonge (1961) with $T = 176$, the Talbot and Patterson method was terminated after 8 CPU sec. Similarly, the CPU times for the $T = 173$ and $T = 176$ problems of Tonge were 3.373 and 1.633 sec, respectively using the Wee and Magazine algorithm. It should also be noted here that Van Assche and Herroelen (1979) experimented with many algorithms differing in their tie-breaking mechanisms and reported the best ones. (We should note here that in the SALB problems it is implicitly assumed that $t_i \leq T$, $\forall\ i \in I$. However, in the original 70-task test problem of Tonge 1961, from an appliance industry, $t_{18} = 319$ whereas $T = 176$. Tonge, as well as, Arcus 1966, Van Assche and Herroelen 1979 and others, handle the situation by allowing the set-up of parallel stations to cover such tasks (i.e., task # 18 requires a parallel station and the idle time in the parallel station, namely, $2(176) - 319 = 33$ time units, can be used for some other task(s)).)

In Table 3 we present a summary from the comparative study of Johnson (1981). These experiments have been performed using different cycle times for the 20-task and 40-task problems of Mastor (1970). We report the mean CPU sec for three different cycle times (from Table 2 of Johnson 1981).

4.5. *Discussion and Remarks*

In general, when the number of tasks is large, all exact algorithms fail, in the sense that the CPU times grow very rapidly. However, some limited success has been attained by Johnson (1981), Wee and Magazine (1981a) and by Talbot and Patterson (1984): Talbot and Patterson, for instance, report that their method, in general, performs well when $m \leq 50$.

As was noted in §3.5, one of the SALBP features affecting the computational performance of the exact algorithms is the number of feasible solutions (sequences; subsets). For instance, the main drawback of the Gutjahr and Nemhauser (1964) method is that the number of nodes generated grows exponentially with problem size: 710 feasible sequences for a 14-task problem and 6320 feasible sequences for a 17-task problem (in 3 CPU sec and 6 CPU min, respectively, on IBM 7090). The same remark is true for the Held, Karp and Sharesian (1963) procedure: the number of "states" is rather large for large problems: 1300 feasible sets for a 36-task problem (in 20 CPU sec on IBM 7090). This problem is true for all DP methods: the computational demands of a DP method grow at an exponential rate with increasing problem size. The Schrage and Baker (1978) labeling procedure and the variations proposed by Kao and Queyranne (1982) are significant developments in this respect. These problems are, of course, not unique to DP methods. For instance, the B&B method of Van

TABLE 3
Computational Comparisons of Johnson (1981) *of Various Methods on the* 20-*Task Problems of Mastor* (1970) *Using IBM* 360/91

Algorithm	CPU Sec
Jackson (1956) (original)	8.01
Jackson (1956) (modified)	3.33
Held, Karp and Sharesian (1963)	9.86+
Gutjahr and Nemhauser (1964)	25.84+
Charlton and Death (1969)	10.35+
Johnson (1981)	3.36

Downloaded from informs.org by [141.161.91.14] on 05 September 2014, at 09:39 . For personal use only, all rights reserved.

Downloaded from informs.org by [141.161.91.14] on 05 September 2014, at 09:39 . For personal use only, all rights reserved.

Assche and Herroelen (1979) has required the generation of 2944, 891 and 2887 nodes, respectively, for $T = 83$, 176 and 355, respectively, for the 70-task problem of Tonge (1961), and the B&B method of Wee and Magazine (1981a) has generated 175 nodes for $T = 176$ using *heuristic* fathoming rules, for the same problem.

The second significant factor limiting the capabilities of the algorithms is the problem of retrieving and storing information during the implementation of the algorithms, including the problem of checking "duplicate" sets/sequences. As the experiments of Held, Karp and Sharesian (1963) and Gutjahr and Nemhauser (1964) and Johnson (1981) demonstrate, a large percentage of the CPU time in the exact algorithms is spent for storing and retrieving information. Thangavelu and Shetty (1971) report that the Geoffrion (1967) method is especially useful in this respect. Johnson (1981) reports that for each algorithm computer core requirements were very roughly proportional to CPU times.

The third factor limiting the capabilities of the exact algorithms is the difficulty of verifying the optimality of a given solution. This is an especially time-consuming task in "newest node search" B&B methods. In any B&B method, optimality is proven only after the implicit or explicit consideration of all the possibilities. The advantage of "newest node search" methods over "frontier search" methods is that the need for either making repeated comparisons between all exposed (live) nodes of the B&B tree or keeping a record of them is eliminated; the feasible solutions are found immediately. Johnson (1981) experimented with the 8 problems of Mastor (1970), each with 20 tasks. He used 3 different cycle times and adjusted each one of the 8 basic problems to have different order strengths, resulting in 72 test problems. He reports that, using his newest node search method, the optimal solutions were found in 0.93, 0.22 and 0.09 mean CPU seconds, respectively, whereas finding and verifying the optimality of these solutions required 23.70, 4.99 and 0.33 CPU seconds, respectively. The disadvantage of "newest node search" methods is that the number of nodes searched is greater than or equal to the number of nodes searched by the "frontier search" methods. It is safe, however, to suggest that "newest node search" methods are more practical for "large" SALB problems: despite the fact that "frontier search" methods require less backtracking, the "newest node search" methods maintain few subproblems on the active list and find a feasible solution early.

The enumerative method of Wee and Magazine (1981a) was affected by the normalized interval ($t_{\min}/T$, $t_{\max}/T$). In fact, enumerative methods are sensitive to the relation between $t_{\max}$ and T: Talbot and Patterson (1984), for instance, note that their algorithm performs well when $T \geq 1.25t_{\max}$ (but not so good otherwise). A similar general remark is not possible, however, for γ: according to the experiments of Johnson (1981), the performance of Jackson (1956), Held, Karp and Sharesian (1963), Gutjahr and Nemhauser (1964), Charlton and Death (1969) and Johnson (1981) were poor when γ exceeded 4 on the test problems of Mastor (1970). Johnson notes that Jackson (1956) (modified), Johnson (1981) and Charlton and Death (1969) stand out as superior to the others. These methods have performed equally well when the $\gamma(P) = 3$ for the 20-task test problems. Johnson notes that the Jackson algorithm is the best when the $\gamma(P) = 2$ and Charlton and Death's algorithm is best when $\gamma(P) \geq 4$. In the 40-task test problems Johnson (1981) is best when $\gamma(P) = 4$. Supremacy of Jackson (1956) over Held, Karp and Sharesian (1963) was thus established by Johnson (1981).

Also note that Thangavelu and Shetty (1971) and Van Assche, Herroelen (1979) and Talbot and Patterson (1984) rely on general IP codes and theory. Charlton and Death (1969) is a general method, applicable to SALBP-1. Jackson (1956), Van Assche and Herroelen (1979) and Wee and Magazine (1981a) have similarities in the enumeration trees that they use. On the other hand, Jackson (1956), Held, Karp and Sharesian (1963) and Johnson (1981) use the concept of maximal stations. On the individual algorithms discussed, we make the following observations: The method of Jackson (1956) is not complete enumeration but explicitly examines all possible assignments to the first $n - 1$ stations before examining any assignment to the nth station. The procedure is, thus, time consuming. The binary search method of Patterson and Albracht (1975), as expected, works well when the bounds are good, that is, when the binary search interval is small. The key to their method is that the lengthy process of verifying optimality is avoided.

As noted before, the Gutjahr and Nemhauser (1964) method results in excessive number of nodes and, in this sense, it is not a practical method. Mansoor (1967) presented an adjustment to the Gutjahr and Nemhauser (1964) algorithm which is capable of finding the optimal solution after considering only a fraction of the "shortest path" calculations. However, as noted by the author, his version of the algorithm may be computationally worse if his backtracking rule has to be used at any stage.

In summary, the methods of Johnson (1981), Wee and Magazine (1981a) and Talbot and Patterson (1984) *appear* to be superior to the other exact algorithms and the studies of Wee and Magazine (1981a) and Talbot and Patterson (1984) appear to establish the supremacy of IP methods over DP methods.

5. Exact Methods for Solving SALBP-2

5.1. *The Algorithms for SALBP-2*

As noted before, SALBP-2 is also $\mathcal{NP}$-hard (Wee and Magazine 1981b) and solving these problems is just as difficult (computationally). There are very few algorithms for solving SALBP-2 because any SALBP-1 method can be used to solve SALBP-2 by successively increasing the cycle time T until a balance is achieved for the stipulated number of stations (see, for instance, Mastor 1970 and Talbot and Patterson 1984). Solution methods for SALBP-2 were first discussed by Helgeson and Birnie (1961). Based on their heuristic method, Helgeson and Birnie proposed an iterative solution procedure. In fact, SALBP-2 methods as a rule iteratively solve a series of SALBP-1 problems.

Almost all of the methods for minimizing the cycle time given a fixed number of stations use (partial) enumeration. The first method of this type was reported by Mansoor (Dar-El)[4] (1964b). This algorithm is based on the heuristic method known as the "ranked positional weights method", due to Helgeson and Birnie (1961). The positional weight of a task is defined, by Helgeson and Birnie, to be the sum of the process time that task and the process times of all of its followers. Helgeson and Birnie rank the tasks in descending order of these weights and, then, assign the tasks to the stations in that order while not violating the precedence and cycle time constraints. The procedure starts with $n_{\min}$ and generates a feasible sequence of tasks which are grouped into subsets (i.e., station assignments). Then the method aims at extending the feasible sequence until all tasks are assigned and grouped into the required number of stations. If a feasible sequence cannot be extended, a backtracking mechanism is used. The last assignment (this is the task with the smallest weight) is cancelled and the procedure is repeated until either the feasible sequence is fully extended (i.e., optimal solution is found) or, if not possible, the (trial) cycle time is raised by one unit. Otherwise, i.e., if the feasible sequence cannot be fully extended, the number of stations must be increased by one. The method is essentially a complete enumeration method.

Mansoor and Yadin (1971) reported another procedure. They begin with the minimum theoretical cycle time as the trial cycle time and sequentially add tasks to

[4] Mansoor and Dar-El are the names of the same person.

Downloaded from informs.org by [141.161.91.14] on 05 September 2014, at 09:39 . For personal use only, all rights reserved.

the admissible subset. The method starts with the empty set. Tasks are added one at a time from the appropriate set of immediate followers (belonging to the subset being extended) until all subsets are complete with respect to the first station (i.e., they are first feasible station assignments). Then the procedure is repeated for second, third stations, etc. If all subsets generated in this way can be extended to include all the tasks for the given cycle time T, then all optimal solutions will be found. Otherwise, the trial cycle time is increased by one unit and the procedure is repeated. The "multiple solutions technique" of Dar-El and Rubinovitch (1979) does exactly what the Mansoor and Yadin method does, except that they generate the entire set—or, 100, often times—of optimal solutions.

A similar method using the positional weights is also given by Rao (1971) called BALB. Using $n_{\min}$, Rao first obtains the theoretical total idle time. He then packs the stations and updates the total idle time. The process is repeated until all stations are packed. If the stations are overpacked, the last assigned task is deleted. If there is no feasible solution, he then increases the cycle time by one.

In a more recent work Wee and Magazine (1981b) modify their B&B algorithm BBX for SALBP-1 (see §4.1.3) and they report four different search methods. Two of the methods search starting with lower and upper bounds, respectively. The other two methods are a "binary search" procedure (BISEARCH) and a "binary and Fibonacci search" procedure (FBS). BISEARCH starts with lower and upper bounds T_L and T_U, respectively, on the optimal cycle time T^* given that $T^*(P, b) = \min \{T | n^*_{P,T} \leq b\}$, where $n^*_{P,T}$ is the minimum number of stations required corresponding to the SALBP-1 problem P, given a fixed cycle time T. At each iteration, the minimum number of stations $n^*(T_L, T_U)$ is found for a cycle time $T_L + (T_U - T_L)/2$. If $n^* > b$, then T becomes the new lower bound T_L. Otherwise, maximum work content of the stations corresponding to T becomes the new upper bound.

The other Wee and Magazine method, namely FBS, starts with T_L and Fibonacci search is applied until a feasible solution is found. Let $T(k)$ be the feasible cycle time obtained in iteration k. The optimal cycle time T^* is then determined by applying BISEARCH in the interval $[T(k-1), T(k)]$.

Charlton and Death (1969) also discuss a procedure for solving SALBP-2: they propose solving the reverse critical path problem subtracting the latest start time of the first task on each machine from the earliest finishing time of the last operation on the machine. Maximum value of this for all the machines provides a lower bound.

5.2. *A Summary of the Reported Computational Experience*

In Table 4 we present a summary of some limited reported computational experience, obtained from Wee and Magazine (1981b). The Wee and Magazine CPU times are for the implementation of their algorithm called FBSBBY which is a binary and Fibonacci search method (variation of their B&B algorithm for SALBP-1 using the heuristic method IUFFD) and both procedures have been implemented by those authors.

5.3. *Discussion and Remarks*

The problem with Mansoor (1964b) is that before the trial cycle time is increased, every feasible sequence must be generated and it is well-known that there simply are too many such sequences. Furthermore, duplicate sequences are generated and, unlike in the Jackson (1956) method, there is no mechanism for checking this. Similar remarks are true for the Mansoor and Yadin (1971) procedure: the list of subsets is very large. Furthermore, as noted by the authors, the number of backtracking operations grow very rapidly as the F-ratio goes up. In fact, most of the methods discussed above require regeneration of sets for each different cycle time when applied

Downloaded from informs.org by [141.161.91.14] on 05 September 2014, at 09:39 . For personal use only, all rights reserved.

TABLE 4

A Summary of Reported Computational Experience

Tasks	Problem Source	Number of Stations	IBM 370/158 CPU Sec Dar-El (1964)	IBM 370/158 CPU Sec Wee and Magazine (1981b)
11	Jackson (1956)	3	0.100	0.033
		4	0.073	0.053
		5	0.116	0.050
21	Mitchell (1957)	3	0.136	0.023
		5	0.136	0.086
		8	0.130	0.051
34	Kilbridge and Wester (1962)	3	0.776	0.770
		6	0.770	0.783
		10	0.600	0.273
70	Tonge (1960)	8	1.251	0.761
		11	1.876	2.666
		21	24.038	16.785

to SALBP-2. The Gutjahr and Nemhauser (1964) method, which can also be used for solving SALBP-2, does not require this effort. However, this method is incapable of handling even modestly sized problems (e.g., with 30–40 tasks). The Dar-El and Rubinovitch's multiple solutions technique, on the other hand, optimally solves 50-task problems with all kinds of F-ratios and problems with up to 140 tasks when the F-ratio is low.

Wee and Magazine (1981b) note that their algorithm BISEARCH requires an excessive number of iterations. Secondly, they state that BISEARCH is efficient only when each cycle time T in the interval $[T_L, T_U]$ is equally likely of being the optimal cycle time. According to Wee and Magazine (1981b), this does not appear to be the case.

We should also point out that the methods of Jackson (1956), Dar-El and Rubinovitch (1979) and Gutjahr and Nemhauser (1964) yield multiple optimal solutions which are useful for line designers who may recursively add more and more constraints until only one or two optimal solutions are generated. (The Jackson and Gutjahr and Nemhauser methods generate all SALBP-2 solutions while solving SALBP-1; the Dar-El and Rubinovitch method attains this by minor changes.) Finally, the SALBP-2 methods generate more efficient solutions (i.e., lower total idle time) than do the methods designed exclusively for SALBP-1. This has some obvious practical advantages.

6. Suggestions for Future Work

ALBP continues to generate interest due to its both practical and theoretical nature as evidenced by recent publications on SALB (e.g., Johnson 1983, Graves and Lamar 1983, Gunther, Johnson and Peterson 1983, Talbot and Patterson 1984) and on related problems (e.g., Ong, Wee and Magazine 1984 and Queyranne 1985). As interest continues, we present some thoughts on future research: *first of all,* there is a *need for new measures for the complexity of SALB problems* (in addition to the measures discussed in §3.5). For instance, the *node-connectivity* of the precedence graph G and the *average degree* in G appear to be relevant measures, as well as *maximum degree.* The latter two may be a better measure than the order strength or the flexibility ratio of the precedence network. Thus, there is a need for some type of composite measures because, for instance, as noted by Wee and Magazine (1981b), the cycle time T by itself is meaningless; it must be viewed with respect to $t_{\min}$ and $t_{\max}$ and, that T should be viewed with respect to t_{ave} and the variance of the task processing times. *Secondly,*

Downloaded from informs.org by [141.161.91.14] on 05 September 2014, at 09:39 . For personal use only, all rights reserved.

there is a *need for new test problems.* In addition to those select problems reported in the literature over the years, some test problems have been generated by Kao and Queyranne (1980), Wee and Magazine (1981a) and Talbot and Patterson (1984). Despite these developments, there definitely is a need for a battery of test problems, featuring differing properties and structures. After a quarter of a century, the 70-task problem of Tonge (1961) remains as the benchmark test problem. *Thirdly,* there is a *need for a new computational comparisons study.* It would be a most useful undertaking if the existing exact algorithms were tested on a common set of problems and, for consistency purposes, by the same implementor(s)/programmer(s). Such a study would be doubly useful if the algorithms were tested for different problem structures. The only existing results of this type are partial, due to Johnson (1981), Wee and Magazine (1981a, b) and Talbot and Patterson (1984). A study of this type on the heuristic/approximate methods has recently been completed by Talbot, Gehrlein and Patterson (1981). *Fourthly,* there is a *definite need for better reporting of computational performance.* The capabilities of the computers vastly differ and, therefore, the CPU times reported without reference to any other computer are not meaningful, except for those readers with knowledge of these machines. Furthermore, whether the reported CPU times are exclusive of input/output times should also be stated clearly.[5]

[5] I thank T. E. Morton (Carnegie-Mellon University) and R. M. V. Rachamadugu (University of Michigan) and J. H. Patterson (Purdue University) for their comments on the earlier version of this paper and I am grateful to the anonymous referees for their constructive suggestions.
I would like to mention the following two new working papers:

JOHNSON, R. V., "Balancing Large Assembly Lines," Graduate School of Management, UCLA, 1985.
BAYBARS, I. AND J. SALTZMAN, "A Two-Process Implicit Enumeration Algorithm for the Simple Assembly Line Balancing Problem," GSIA WP # 29-85-86, Carnegie-Mellon University, February 1986.

Johnson (1985) proposes a 'laser' type newest node B&B algorithm, with logic designed for very fast achievement of feasibility, ensuring a feasible solution to any line of 1000 or even more tasks.

References

ARCUS, A. L., "COMSOAL—A Computer Method of Sequencing Operations for Assembly Lines," *Internat. J. Production Res.,* 4 (1966), 259–277.
BALAS, E., "An Additive Algorithm for Solving Linear Programs with 0-1 Variables," *Oper. Res.,* 13 (1965), 517–546.
——— AND E. ZEMEL, "An Algorithm for Large Zero-One Knapsack Problems," *Oper. Res.,* 28 (1980), 1130–1154.
BAKER, K., *Introduction to Sequencing and Scheduling,* Wiley, New York, 1974.
BAYBARS, I., "Optimal Assignment of Broadcast Frequencies," *European J. Oper. Res.,* 9 (1982), 257–263.
———, "A Survey of Inexact Algorithms for the Simple Assembly Line Balancing Problem," GSIA WP-86-82-83, Carnegie-Mellon University, Pittsburgh, June 1984.
———, on "Currently Practiced Formulations of the Assembly Line Balance Problem," *J. Oper. Management,* 5 (1985), 449–453.
——— and A. Frieze, "Expected Behavior of Line Balancing Heuristics," *IMA J. Math. in Management,* (1986), (to appear).
BOWMAN, E. H., "Assembly Line Balancing by Linear Programming," *Oper. Res.,* 8 (1960), 385–389.
BRYTON, B., "Balancing of a Continuous Production Line," Unpublished M.S. Thesis, Northwestern University, Evanston, ILL., 1954.
CHARLTON, J. M. AND C. C. DEATH, "A General Method for Machine Scheduling," *Internat. J. Production Res.,* 7 (1969), 207–217.
DAR-EL, E. M., "MALB-A Heuristic Technique for Balancing Large Single-Model Assembly Lines," *AIIE Trans.,* 5 (1973), 343–356.
———, "Mixed-Model Assembly Line Sequencing Problems," *Omega,* 6 (1978), 317–323.
——— AND R. F. COTHER, "Assembly Line Sequencing for Model Mix", *Internat. J. Production Res.,* 13 (1975), 463–477.
——— AND Y. RUBINOVITCH, "MUST—A Multiple Solutions Technique for Balancing Single Model Assembly Lines," *Management Sci.,* 25 (1979), 1105–1114.

Downloaded from informs.org by [141.161.91.14] on 05 September 2014, at 09:39 . For personal use only, all rights reserved.

Downloaded from informs.org by [141.161.91.14] on 05 September 2014, at 09:39 . For personal use only, all rights reserved.

DENARDO, E. V. AND B. L. FOX, "Shortest Route Methods. I: Reaching, Pruning and Buckets", *Oper. Res.*, 27 (1979), 161–186.

FREEMAN, D. R., "A General Line Balancing Model," *Proc. 19th Annual Conf. AIIE*, Tampa, FLA., 1968, 230–235.

——— AND J. V. JUCKER, "The Line Balancing Problem," *J. Industrial Engineering*, 18 (1967), 361–364.

GAREY, M. B. AND D. S. JOHNSON, *Computers and Intractability*, Freeman, San Francisco, 1978.

——— AND ———, "Approximate Algorithms for the Bin Packing Problem: A Survey," in *Analysis and Design of Algorithms in Combinatorial Optimization*, G. Auselio and M. Lucertini, (eds.), Springer, New York, 1981, 147–172.

———, R. L. GRAHAM, D. S. JOHNSON AND A. C. YAO, "Multiprocessor Scheduling as Generalized Bin Packing," *J. Combinatorial Theory*, 21 (1976), 257–298.

GEOFFRION, A. M., "Integer Programming by Implicit Enumeration and Balas' Method," *SIAM Rev.*, 9 (1967), 178–190.

GRAVES, S. C. AND B. W. LAMAR, "An Integer Programming Procedure for Assembly System Design Problems," *Oper. Res.*, 31 (1983), 522–545.

GROOVER, M. P., *Automation, Production Systems and Computer-Aided Manufacturing*, Prentice-Hall, Englewood Cliffs, NJ., 1980.

GUNTHER, R. E., G. D. JOHNSON AND R. S. PETERSON, "Currently Practiced Formulations for the Assembly Line Balance Problem", *J. Oper. Management*, 3 (1983), 209–221.

GUTJAHR, A. L. AND G. L. NEMHAUSER, "An Algorithm for the Line Balancing Problem," *Management Sci.*, 11 (1964), 308–315.

HELD, M. AND R. M. KARP, "A Dynamic Programming Approach to Sequencing Problems," *SIAM*, 10 (1962), 196–210.

———, ——— AND R. SHARESIAN, "Assembly Line Balancing–Dynamic Programming with Precedence Constraints", *Oper. Res.*, 11 (1963), 442–459.

HELGESON, W. B. AND D. P. BIRNIE, "Assembly Line Balancing Using Ranked Positional Weight Technique," *J. Industrial Engineering*, 12 (1961), 394–398.

JACKSON, J. R., "A Computing Procedure for a Line Balancing Problem," *Management Sci.*, 2 (1956), 261–271.

JAESCHKE, G., "Eine Allgemeinen Methode Zur Losung Kombinatorisher Probleme," *Ablauf- und Planungforschung*, 5 (1964), 133–153.

JOHNSON, R. V., "Branch and Bound Algorithms for Assembly Line Balancing and Job-Shop Scheduling", Unpublished Ph.D. Thesis, University of California, Los Angeles, 1973.

———, "Assembly Line Balancing Algorithms: Computation Comparisons," *Internat. J. Production Res.*, 19 (1981), 277–287.

———, "A Branch and Bound Algorithm for Assembly Line Balancing Problems with Formulation Irregularities", *Management Sci.*, 29 (1983), 1309–1324.

KAO, E., "Computational Experience wth a Stochastic Assembly Line Balancing Algorithm," *Computers and Oper. Res.*, 6 (1979), 79–86.

KAO, P. C. AND M. QUEYRANNE, "240 Assembly Line Balancing Problems," Working Paper, University of Houston, Houston, 1980.

——— AND ———, "On Dynamic Programming Methods for Assembly Line Balancing," *Oper. Res.*, 30 (1982), 375–390.

——— AND ———, "Hybrid Dynamic Programming/Branch and Bound Algorithms for Assembly Line Balancing," Presented at the ORSA/TIMS Joint National Conference, Orlando, FLA., November 1983.

KARP, R. M., "Reducibility Among Combinatorial Problems", in *Complexity of Computer Applications*, R. E. Miller and J. W. Thatcher (eds.), Plenum, New York, 1972, 85–104.

KILBRIDGE, M. D. AND L. WESTER, "The Balance Delay Problem," *Management Sci.*, 8 (1961), 69–84.

——— AND ———, "A Review of Analytical Systems of Line Balancing," *Oper. Res.*, 10 (1962), 626–638.

KLEIN, M., "On Assembly Line Balancing," *Oper. Res.*, 11 (1963), 274–281.

LAWLER, E. L., "Efficient Implementation of Dynamic Programming Algorithms for Sequencing Problems," Report BW 106/79, Stichting Mathematisch Centrum, Amsterdam, 1979.

LIAS, E. J., "Tracking the Elusive KOPS," *Datamation*, (1980), 99–105.

MACASKILL, J. L. C., "Production Line Balances for Mixed Model Lines," *Management Sci.*, 19 (1972), 423–434.

MANSOOR, E. M., "Assembly Line Balancing—An Improvement on the Ranked Positional Weight Technique," *J. Industrial Engineering*, 15 (1964a), 73–78.

———, "Assembly Line Balancing—Extensions and Discussion," *J. Industrial Engineering*, 15 (1964b), 322–323.

———, "Improvement on the Gutjahr and Nemhauser Algorithm for Line Balancing Problems," *Management Sci.*, 14 (1967), 250–254.

RIGHTSLINK

——— AND M. YADIN, "On the Problem of Assembly Line Balancing," in *Developments in Operations Research,* B. Avi-Ithzak (ed.), Gordon and Breach, New York, 1971, 361–375.

MASTOR, A. A., "An Experimental Investigation and Comparative Evaluation of Production Line Balancing Techniques," *Management Sci.,* 16 (1970), 728–746.

MERTENS, P., "Assembly Line Balancing by Partial Enumeration," *Ablauf- und Planungforschung,* 8 (1967), 429–433.

MITCHELL, J., "Computational Procedure for Balancing Zoned Assembly Lines," Research Report 6-94801-1-R3, Westinghouse Research Laboratories, Pittsburgh, 1957.

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

NANDA, R. AND J. M. SCHER, "Non-parallelability Constraints in Assembly Lines with Overlapping Work Stations," *AIIE Trans.,* 8 (1976), 343–349.

ONG, H. L., T. S. WEE AND M. J. MAGAZINE, "Probabilistic Analysis of Bin Packing Heuristics," *Oper. Res.,* 32 (1984), 983–998.

PATTERSON, J. H. AND J. J. ALBRACHT, "Assembly Line Balancing: 0-1 Programming with Fibonacci Search," *Oper. Res.,* 23 (1975), 166–174.

PINTO, P. A., D. G. DANNENBRING AND B. M. KHUMAWALA, "A Branch and Bound Algorithm for Assembly Line Balancing with Paralleling," *Internat. J. Production Res.,* 13 (1975), 183–196.

———, ——— AND ———, "A Heuristic Network Procedure for the Assembly Line Balancing", *Naval Res. Logist. Quart.,* 25 (1978), 299–307.

———, ——— AND ———, "Branch and Bound and Heuristic Procedures for Assembly Line Balancing with Parallel Stations," *Internat. J. Production Res.,* 19 (1981), 565–576.

———, ——— AND ———, "Assembly Line Balancing with Processing Alternatives: An Application," *Management Sci.,* 29 (1983), 817–830.

PRENTING, T. O. AND N. T. THOMOPOULOS, *Humanism and Technology in Assembly Line Systems,* Hayden, Rochelle Park, NJ., 1974.

QUEYRANNE, M., "Bounds for Assembly Line Balancing Heuristics," *Oper. Res.,* 33 (1985), 1353–1359.

RAO, D., "Single and Mixed Model Assembly Line Balancing Methods for Both Deterministic and Normally Distributed Work Element Times," M.S. Thesis, Oregon University, 1971.

SALVESON, M. E., "The Assembly Line Balancing Problem," *J. Industrial Engineering,* 6 (1955), 18–25.

SCHRAGE, L. AND K. R. BAKER, "Dynamic Programming Solution of Sequencing Problems with Precedence Constraints," *Oper. Res.,* 26 (1978), 444–459.

SPHICAS, G. P. AND F. N. SILVERMAN, "Deterministic Equivalents for Stochastic Assembly Line Balancing," *AIIE Trans.,* 8 (1976), 280–282.

TALBOT, F. B., W. V. GEHRLEIN AND J. H. PATTERSON, "A Comparative Evaluation of Heuristic Line Balancing Techniques," Working Paper # 215, GBS, University of Michigan, Ann Arbor, 1981.

——— AND J. H. PATTERSON, "An Integer Programming Algorithm with Network Cuts for Solving the Single Model Assembly Line Balancing Problem," *Management Sci.,* 30 (1984), 85–99.

THANGAVELU, S. R. AND C. M. SHETTY, "Assembly Line Balancing by Zero-One Integer Programming," *AIIE Trans.,* 3 (1971), 61–68.

THOMOPOULOS, N. T., "Line Balancing—Sequencing for Mixed Model Assembly," *Management Sci.,* 14 (1967), B59–B75.

TONGE, F. M., *A Heuristic Program for Assembly Line Balancing,* Prentice-Hall, Englewood Cliffs, NJ., 1961.

VAN ASSCHE, F. AND W. S. HERROELEN, "An Optimal Procedure for the Single-Model Deterministic Assembly Line Balancing Problem," *European J. Oper. Res.,* 3 (1979), 142–149.

VRAT, P., "Cost Model for Optimal Mix of Balanced Stochastic Assembly Line and Modular Assembly System for Customer Oriented Production System," *Inter. J. Production Res.,* 14 (1976), 445–463.

WEE, T. S. AND M. J. MAGAZINE, "An Efficient Branch and Bound Algorithm for Assembly Line Balancing—Part 1: Minimize the Number of Work Stations," Working Paper # 150, University of Waterloo, Waterloo, ONT., 1981a.

——— AND ———, "An Efficient Branch and Bound Algorithm for Assembly Line Balancing—Part 2: Maximize the Production Rate," Working Paper # 151, University of Waterloo, Waterloo, ONT., 1981b.

——— AND ———, "Assembly Line Balancing as Generalized Bin Packing," *Oper. Res. Lett.,* 1 (1982), 56–58.

WHITE, W. W., "Comments on a Paper by Bowman," *Oper. Res.,* 9 (1961), 274–276.

Downloaded from informs.org by [141.161.91.14] on 05 September 2014, at 09:39 . For personal use only, all rights reserved.

Copyright 1986, by INFORMS, all rights reserved. Copyright of Management Science is the property of INFORMS: Institute for Operations Research and its content may not be copied or emailed to multiple sites or posted to a listserv without the copyright holder's express written permission. However, users may print, download, or email articles for individual use.

Downloaded from informs.org by [141.161.91.14] on 05 September 2014, at 09:39 . For personal use only, all rights reserved.

RIGHTSLINK