Pergamon

PII: S0360-8352(96)00301-4

Computers ind. Engng Vol. 32, No. 2, pp. 321–332, 1997

0360-8352/97 $17.00 + 0.00

A HEURISTIC FOR AN ASSEMBLY LINE BALANCING PROBLEM WITH INCOMPATIBILITY, RANGE, AND PARTIAL PRECEDENCE CONSTRAINTS

KYUNGCHUL PARK,[1] SUNGSOO PARK[1]* and WANHEE KIM[2]

[1]Department of Industrial Engineering, Korea Advanced Institute of Science and Technology, Taejon 305-701, Korea

[2]Samsung Data Systems, Seoul, Korea

(Received 24 September 1996)

Abstract—This paper considers a new type of an assembly line balancing problem which occurs in a company manufacturing electronic home appliances. The problem has some special characteristics. The main difficulty in this problem is that the precedence diagram is not enough to describe the precedence relationships between the tasks. It is shown that the usual line balancing model is a special case of the problem considered in this paper. After considering two subproblems, a heuristic algorithm for the problem is proposed. The algorithm is based on some network theories. Computational results based on some real world problems show the proposed algorithm works well in practice. A full system which incorporates graphic user interface is developed and currently in use. © 1997 Elsevier Science Ltd. All rights reserved

1. INTRODUCTION

In recent years, many OR/MS techniques have been developed and successfully used to solve decision making problems. However, it is felt by some practitioners that the well-established models and methodologies found in the OR/MS literature are rather inadequate to be used in their applications since they cannot reflect many factors arising in the real world. This paper presents an experience to solve a real world problem using well known OR/MS techniques.

One of the extensively studied problems in production management is the line balancing problem. Usually, the problem is stated as follows [1]. For a given set of tasks with their processing times and a set of workstations, find an assignment of each task to a workstation to minimize the cycle time while satisfying the precedence constraints imposed on the set of tasks. Many useful techniques have been proposed to solve the model. For example, see [2] for a survey of the exact algorithms and [3] for a survey of heuristics. Recently, Johnson [4] and Hoffman [5] proposed efficient search procedures for the problem.

Though the model as stated above is very useful in many situations, in practice, there may exist more complicated constraints. For example, some pairs of tasks cannot be assigned into the same workstation because of incompatibility between them caused by the different characteristics of the operations needed, the materials used, the limited workspace, or other technological factors. Also, there can exist a natural distinction of the tasks according to the process design. Johnson [6] considered the line balancing problem with a few modifications, such as preplanned imbalance and assigning tasks to particular types of workstations.

The line balancing problem (LBP) considered in this paper occurs in a company manufacturing home appliances and has many characteristics different from the usual line balancing models. This paper proposes a heuristic procedure to solve the problem.

This paper is structured as follows. In Section 2, the statement of the problem is presented with a few preliminary analyses. Section 3 presents two subproblems which will be used to develop a solution algorithm of the line balancing problem. Section 4 presents analysis of the two subproblems and Sections 5 and 6 give algorithms to solve them. A heuristic procedure for the

*To whom all correspondence should be addressed.

line balancing problem is presented in Section 7 and computational results are given in Section 8. Finally, Section 9 gives concluding remarks.

2. STATEMENT OF THE PROBLEM

The line balancing problem (LBP) considered in this paper has the following characteristics.

(i) The set of tasks is partitioned into two disjoint sets. One is called the set of *fixed tasks*, and the other is the set of *float tasks*.
(ii) On the set of fixed tasks, a linear precedence is imposed, that is, the fixed *tasks* should be performed one after the other.
(iii) Some pair of consecutive fixed tasks cannot be assigned to the same workstation because of the incompatibility between them.
(iv) For each float task, there is a set of ranges expressed in terms of pairs of fixed tasks. The float task should be assigned between the fixed tasks specified in any one of the ranges.
(v) For some pairs of float tasks, there exist precedence constraints.

Figure 1 illustrates an example of the line balancing problem. The determination of the characteristic of a task as a fixed or a float task is made in the process design. The fixed tasks can be thought of as the primary operations needed to assemble a given product, which usually have a natural sequence. The set of float tasks can be considered as the secondary operations. The distinction is somewhat flexible in the sense that some tasks may be defined as fixed or floating depending on the situation.

Let us call the constraints in (iii) the *incompatibility constraints*, (iv) the *range constraints*, and (v) as the *partial precedence constraints*. The problem considered in this paper is to find an assignment of the tasks to the workstations, which minimizes the cycle time and satisfies the above mentioned constraints while the number of workstations is given.

The main difficulty encountered when applying the existent line balancing methods to this problem is that the precedence diagram is not enough to describe the range constraints. The number of ranges attached to each float task is usually greater than one. For example in Fig. 1, a float task numbered 1 can be inserted between the fixed tasks numbered 1 and 3, or between the fixed tasks numbered 6 and 7. This lack of consecutiveness comes from the compatibility condition of the tasks.

It can be shown that the usual line balancing problem can be considered as a special case of the current problem. Suppose an instance of a usual line balancing problem is given with a precedence diagram G (for example, see Fig. 2). Further, let 0 and m be the dummy tasks

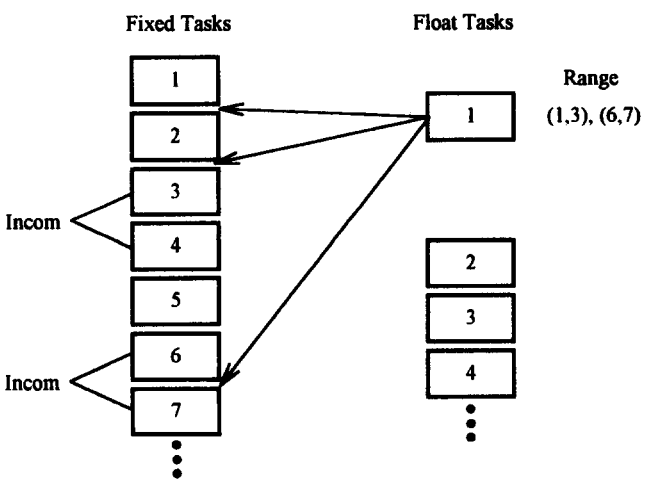


Fig. 1. An example of the line balancing problem.

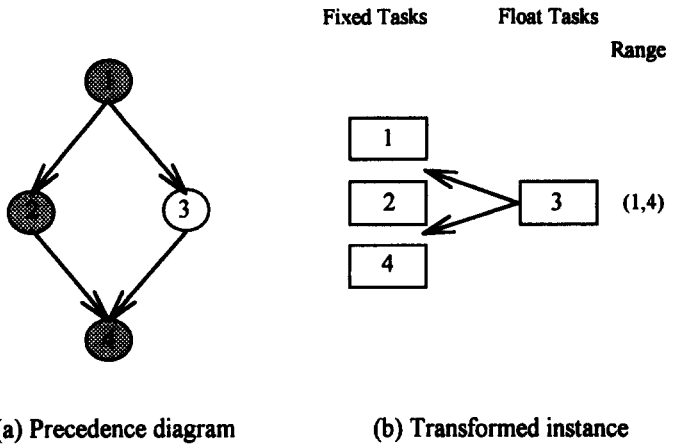


Fig. 2. Transformation of usual line balancing problem into LBP.

representing the start and end of assembly, respectively. Since the graph G is acyclic, the longest $(0, m)$-path (in terms of the number of edges) in G can be computed easily. Let the set of tasks on the path be M, which is viewed as a set of fixed tasks. For example, in Fig. 2, $M = \{1,2,4\}$. Now for each task not in M (that can be thought as a float task), it is easy to define the set of ranges in terms of fixed tasks where it can be assigned. Note that, in this case, the set of ranges for each float task consists of exactly one range. Otherwise, there should exist a cycle in G. Now by defining the remaining precedence constraints on the set of float tasks as specified in the original precedence diagram, an instance of the current line balancing problem can be obtained.

In this paper, an efficient heuristic procedure based on the neighborhood search technique is presented. For a given feasible solution, an improved solution is generated by solving two subproblems. The first subproblem aims at redistributing the float tasks while preserving a given assignment of the fixed tasks. The second subproblem aims at reoptimizing the assignment of the fixed tasks while preserving the current relative sequence of the float tasks. More explanations on the subproblems will be given in the following sections. Procedures to solve each subproblem are proposed, which are based on network theories. Then a solution procedure for (LBP) is developed, which iteratively calls the algorithms for the two subproblems.

3. TWO SUBPROBLEMS

This section presents some notations which will be used later. Then two subproblems of (LBP) which are used in developing the solution procedure are presented. For simplicity of presentation, the partial precedence constraints on the set of float tasks will be ignored until Section 7.

3.1. Notations

$M = \{1, \ldots, m\}$: the set of fixed tasks
$N = \{1, \ldots, n\}$: the set of float tasks
$W = \{1, \ldots, w\}$: the set of workstations
p_i: the processing time of the fixed task $i \in M$
a_j: the processing time of the float task $j \in N$
C: current cycle time

Given a feasible assignment of the fixed tasks, the following notations are used.

$W(k)$: the set of fixed tasks assigned to the workstation $k \in W$
$B(j)$: the set of workstations to which the float task j can be assigned, for $j \in N$
$A(k)$: the set of float tasks which can be assigned to the workstation $k \in W$

Note that $B(j)$ and $A(k)$ can be found using the range constraints and the given assignment of the fixed tasks.

3.2. Subproblem 1

For a given feasible solution of (LBP), subproblem 1 (Sub1) is used to find an improved solution by changing the assignment of float tasks while preserving the current assignment of fixed tasks. The objective of (Sub1) is to determine whether the float tasks can be assigned into the workstations within the predetermined cycle time C while a feasible assignment of the fixed tasks is given. Here, feasibility implies that the assignment of the fixed tasks satisfies the incompatibility constraints. The number of workstations is also predetermined. Note that by iteratively solving (Sub1) with varying the cycle time C, the assignment of float tasks that minimizes the cycle time can be found.

3.3. Subproblem 2

Similarly to (Sub1), for a given feasible solution of (LBP), subproblem 2 (Sub2) is used to find an improved solution by changing the assignment of the fixed and float tasks while preserving the relative order on the float tasks present in the given solution. (Sub2) is to find an assignment of both fixed and float tasks to workstations which minimizes the number of workstations needed while a predetermined cycle time C is given. Here a linear precedence on all the float tasks is additionally imposed. The linear precedence specifies the order in which the float tasks are performed.

4. ANALYSIS OF THE TWO SUBPROBLEMS

This section presents analysis on the two subproblems presented in Section 3.

4.1. Subproblem 1

(Sub1) can be viewed as a generalized bin packing problem where the set of workstations corresponds to the set of bins and the set of float tasks corresponds to the set of items to be inserted. Since the bin packing problem is NP-complete [7], (Sub1) is NP-complete in general. The bins (workstations) in which each item (float task) can be inserted are restricted depending on the range constraints. In addition, the sizes of bins are not uniform since they depend on the given assignment of the fixed tasks. Many heuristics have been proposed for the bin packing problem, and for some of them worst case performance bounds have been derived. See Garey and Johnson [7], for details. Friesen and Langston [8] analyse algorithms for a problem with different bin sizes. However, these algorithms cannot be applied to (Sub1) since there exist restrictions on the bins for an item to be inserted. In this section, a new algorithm based on the transportation problem is presented.

The following problem is a (continuous) relaxation of (Sub1).

$$
\begin{aligned}
\text{(TP)} \quad \min \quad & \sum_{j=1}^{n} \sum_{k \in B(j)} x_{jk} \\
s.t. \quad & \sum_{k=1}^{w} x_{jk} = a_j && \text{for all } j \in N \\
& \sum_{j=1}^{n} x_{jk} + s_k = b_k && \text{for all } k \in W \\
& x_{jk} \geqslant 0 && \text{for all } j \in N,\ k \in W \\
& s_k \geqslant 0 && \text{for all } k \in W,
\end{aligned}
$$

where $b_k = C - \sum_{i \in W(k)} p_i$.

Note that the above problem can be viewed as a transportation problem (TP). For an example of (TP), see Fig. 3. In Fig. 3, the assignment of the fixed tasks is given and from this, the set of workstations to which each float task can be assigned is determined. For example, for the float task 1, since the range attached to it is (1,10), it can be assigned to the workstation 1, 2, or 3, where at least one fixed task in the range is assigned. The bold lines represent valid assignments and dotted lines represent invalid assignments, that is, assignments to the workstations at which the float task cannot be done.

In (TP), x_{jk} represents the amount of the processing time of the float task j allocated to the workstation k and s_ks are slack variables. The objective function represents the amount of the processing time allocated to the invalid workstations, that is, the workstations to which the float task cannot be assigned. Therefore (Sub1) is to find a solution to (TP) with the objective value 0. Such a solution should minimize the objective function of (TP), since for any feasible solution to (TP), the objective value is non-negative.

If (Sub1) has a feasible solution, then the solution is an optimal solution of (TP) and the optimal objective value is 0. On the other hand, if (TP) has an optimal solution whose objective value is greater than 0, (Sub1) is infeasible. So a necessary condition for (Sub1) to be feasible is that (TP) has an optimal solution whose objective value is 0. Suppose the necessary condition holds. If the optimal solution gives an actual assignment of the float tasks to the workstations, it is a feasible solution of (Sub1). The actual assignment can be deduced from the solution only when for each j, x_{jk}s are all zero except for one k. Then the float task j can be assigned to the workstation k. This case occurs when there exists an optimal basic feasible solution of (TP) which has all the slack variables as basic variables. The following proposition summarizes the above remarks.

PROPOSITION 1.

(1) *If* (Sub1) *is feasible, then the optimal objective value of* (TP) *is* 0.

(2) *If the optimal objective value of* (TP) *is greater than* 0, *then* (Sub1) *is infeasible.*

(3) *The necessary and sufficient condition for* (Sub1) *to be feasible is that the optimal value of* (TP) *is* 0 *and there exists an optimal basic solution which has all the slack variables as basic variables.*

Proof. (1) and (2) are clear from the above discussion. Only (3) needs to be proved here. Suppose the optimal objective value of (TP) is 0 and there exists an optimal basic solution which has all

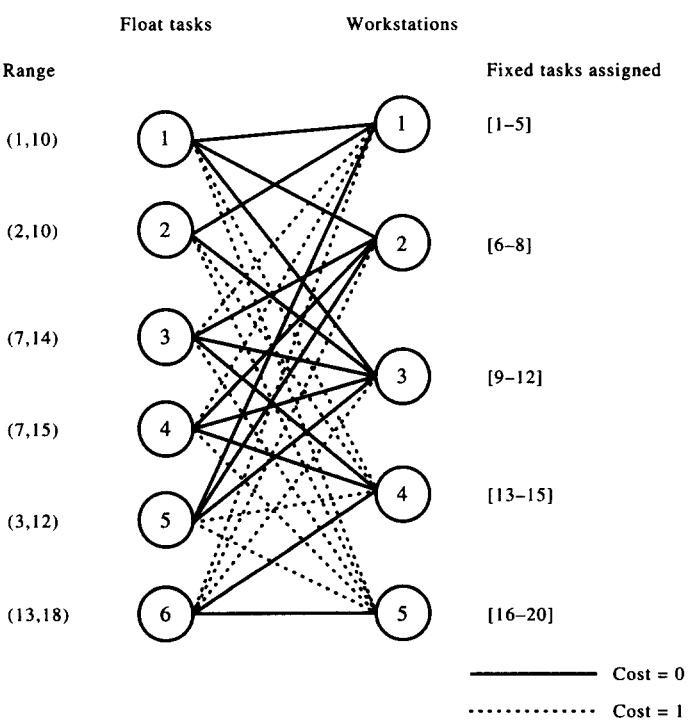


Fig. 3. Construction of (TP).

the slack variables as basic variables. Any basic solution of (TP) corresponds to a spanning tree of the (complete) bipartite graph induced by $N \cup \{s\}$ and W, where s is the slack node. So the number of basic variables is $n + w$. Since there are w slack variables and all of them are basic, only n of the variables x_{jk} are basic. Hence only one of the variables x_{jk} for each j should be basic. Using the solution, the actual assignment can be obtained. On the other hand, if (Sub1) is feasible, the variables corresponding to the feasible assignment together with all the slack variables give an optimal basic solution to (TP).

From (3) in Proposition 1, it can be deduced that (Sub1) is equivalent to the problem of finding an optimal basic solution whose objective value is 0 and where the slack variables are basic variables in the greatest possible extent. Not surprisingly, the latter problem is also NP-complete [9]. Therefore, it is unlikely that there exists a polynomial time algorithm to solve it. In the next section, a heuristic procedure to solve the problem is presented, which is based on the concept of augmenting paths.

4.2. Subproblem 2

Like (Sub1), (Sub2) can also be viewed as a type of bin packing problem. In this case, the set of both the fixed tasks and the float tasks is the set of items to be inserted. However, in contrast to (Sub1), (Sub2) can be solved in polynomial time by a shortest path algorithm.

Let us assume that by renumbering the task numbers, if necessary, the set of float tasks N is $\{1, \ldots, n\}$ and a tentative linear order is given from the task 1 to n. Let us construct a directed graph $G = (V, A)$ as follows. Let V be given as the set $\{[ij] | i \in M, j \in N\} \cup \{[0,0]\}$. An arc $([i_1,j_1],[i_2,j_2])$ exists if, and only if, the set of the tasks $\{i_1 + 1, \ldots, i_2\} \cup \{j_1 + 1, \ldots, j_2\}$ can be assigned to the same workstation while neither exceeding the cycle time C nor violating the incompatibility constraints. In the preceding remark, if $i_1 = i_2$ or $j_1 = j_2$, the corresponding set should be interpreted as empty.

Now suppose a shortest path from the node [0,0] to the node $[m,n]$ be given. Let the path be given as $[0,0] \rightarrow [i_1,j_1] \rightarrow \ldots \rightarrow [i_p,j_p]$, where $[i_p,j_p] = [m,n]$. Then if the set of tasks $\{i_{t-1} + 1, \ldots, i_t\} \cup \{j_{t-1} + 1, \ldots, j_t\}$ is assigned to the workstation t, the assignment gives an optimal solution to (Sub2). Since the number of the nodes is mn, the shortest path can be found in polynomial time.

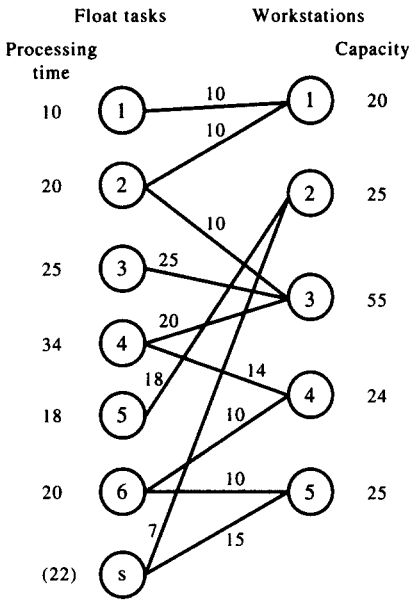
PROPOSITION 2. (Sub2) *can be solved in polynomial time.*

Though the optimal assignment can be found by solving the corresponding shortest path problem, it is not practical to construct the graph G beforehand and then use the shortest path algorithm on G since a large memory is needed to store the information of G in a computer. In the next section, an algorithm is presented which successively generates the necessary part of the graph as the algorithm progresses.

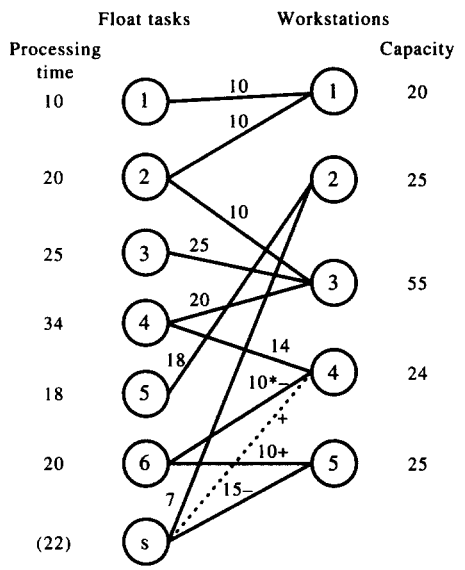
5. SOLUTION METHOD FOR THE SUBPROBLEM 1

As noted in Section 4.1, (Sub1) is equivalent to the problem of finding an optimal basic solution of (TP) whose objective value is 0 and which has slack variables as basic variables as many as possible. (Sub1) is feasible only when the resulting solution of (TP) has all the slack variables as basic variables. Let us call an optimal basic solution of (TP) an improvement over the other optimal basic solution, if the number of basic slack variables in the former is greater than that in the latter. In this section, a heuristic algorithm (called *slack pivoting procedure*) is presented which improves the current optimal basic solution of (TP) by pivoting the non-basic slack variables into the basis if possible.

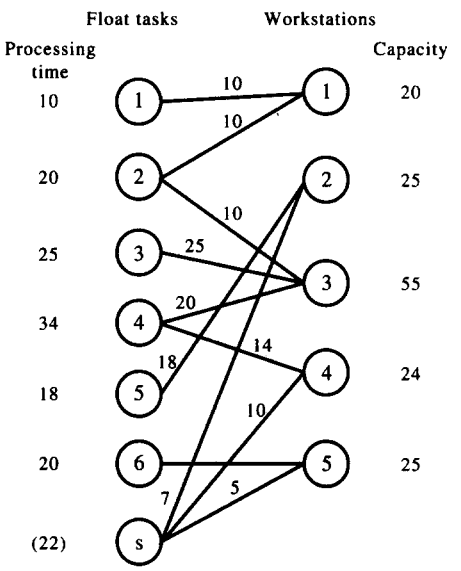
If (TP) has the optimal value greater than 0, (Sub1) is infeasible. Therefore, in the following, it is assumed that the optimal value of (TP) is 0 and an optimal basic solution (x^1,s^1) is given. (TP) can be solved by any standard algorithm for the transportation problem (e.g. see [10]). If all the slack variables are basic variables, then a feasible assignment can be found. Hence let us assume that there exists at least one non-basic slack varable, say s_j. If the variable s_j is pivoted into the basis, there occurs a unique cycle in the subgraph of G induced by the arcs corresponding to basic variables [see Fig. 4(b)]. Let the cycle be D. Let us label each arc alternately by the signs + and − starting from the arc corresponding to s_j which is labeled + [see Fig. 4(b)]. The signs on the



(a) An optimal solution of (TP)



(b) Labeling



(c) An improved solution found

Fig. 4. Slack pivoting procedure.

arcs in the cycle indicate whether the flows on each arc should be increased or decreased to conserve the flows on the nodes when the flow on s_j is increased. Note that an improvement upon the current solution is possible if min $\{x_{ij}^1, s_k^1 | (i,j),(s,k) \in D$ and labeled by the sign $-\}$ is attained at the x variable [e.g., x_{64} in Fig. 4(b)]. If this happens, s_j can be pivoted into the basis, which results in an improved solution (x^2, s^2) [see Fig. 4(c)]. If not, the procedure fails and then try another non-basic slack variable, if any. Note that the pivot is not performed if it increases the objective value. The procedure is continued on all non-basic slack variables until no further improvement is possible.

At the end of the procedure, if the final solution has all the slack variables as basic, (Sub1) is solved. Otherwise, try to construct a feasible solution of (Sub1) from the final solution of (TP) using a heuristic procedure. The procedure is described as follows.

5.1. Feasible solution construction procedure (FSCP)

Suppose the final solution obtained at the end of the slack pivoting procedure is given as (x^*,s^*) and there exists at least one float task whose processing time is divided and allocated to at least two different workstations. Let P be the set of such float tasks. (FSCP) determines an actual assignment of the float tasks in P.

First, a proposition on the cardinality of P is given.

PROPOSITION 3. $|P| \leqslant w$

Proof. The number of basic variables in (TP) is $n + w$. Let p be the cardinality of P. Then (number of the basic variables) $= n + w \geqslant 2p + (n - p) = n + p$. Hence the relation holds.

Now consider the following problem (MP(C)).

$$\text{(MP(C))} \quad \min \max_k \sum_{j \in P} w_{jk} y_{jk}$$

$$\sum_{k \in B(j)} y_{jk} = 1 \qquad \text{for all } j \in P$$

$$\sum_{A(k) \cap p} y_{jk} \leqslant 1 \qquad \text{for all } k \in W$$

$$y_{jk} \in \{0,1\} \qquad \text{for all } j \in P,\ k \in W$$

where $w_{jk} = \max(0, a_j + \sum_{i \in Q(k)} a_i - C)$ and $Q(k) = W(k) \cup \{i \in N \backslash P | x_{ik} = a_i\}$.

Note that $W(k)$ is the set of the fixed and float tasks which are already assigned to the workstation k. w_{jk} is the amount of increment over the current cycle time C if the float task j is assigned to the workstation k.

(MP(C)) determines the assignment of the float tasks in P to the workstations so that the maximum increment over the current cycle time C is minimized. It can be viewed as a minmax matching problem and can be solved efficiently [11]. The following proposition shows that (MP(C)) always has a feasible solution.

PROPOSITION 4. (MP(C)) always has a feasible solution.

Proof. Can be proved by using Proposition 3 and the Hall's theorem [11]. The proof is omitted here.

If the optimal objective value of (MP(C)) is 0, the optimal solution of (MP(C)) provides a feasible solution of (Sub1). Although this may not happen, the procedure can be used to obtain a solution which is close to a feasible solution.

6. SOLUTION METHOD FOR THE SUBPROBLEM 2

As noted in Section 4.2, (Sub2) can be solved in polynomial time by the shortest path algorithm. However, as noted earlier, large computer memory is needed to store the graph explicitly. So instead of generating the graph completely beforehand, only the needed portion of the graph is generated as the algorithm progresses. Note that Dijkstra's algorithm for the shortest path problem [11] can be implemented without storing the graph completely if the information needed can be supplied at the various stages of the algorithm (by any method). Since the graph G has a special structure (it does not have a cycle), it is easy to construct the set of nodes which are adjacent

to the set of nodes currently considered. More storage saving is possible if only the maximal nodes are generated. Here, the maximality means the following. Suppose the node considered currently is $[i,j]$. Then the node $[i',j']$, where $i' \geqslant i, j' \geqslant j$, is maximal with respect to $[i,j]$ if the set of the tasks $T = \{i+1, \ldots, i'\} \cup \{j+1, \ldots, j'\}$ can be assigned to the same workstation without violating the incompatibility constraint, but neither $T \cup \{i'+1\}$ nor $T \cup \{j'+1\}$ can have such property.

Hence by using the above scheme, (Sub2) can be solved with less memory requirements.

7. THE SOLUTION ALGORITHM

By combining the solution algorithms for (Sub1) and (Sub2), a heuristic algorithm for the line balancing problem is developed. The following is the procedure.

7.1. Line balancing heuristic(LBH)

(*Step* 0) Get an initial solution. Let the resulting cycle time be C_0. $k = 1$.

(*Step k*)

1. *Phase I*
   - *i. Set* $C_U = C_{k-1}$. Fix the assignment of the fixed tasks as the solution at step $k-1$.
   - ii. Using the binary search technique on the cycle time $C \in [C_L, C_U]$, find the minimum C^* such that (TP) has an optimal solution whose objective is 0. If all the slacks are basic, go to v.
   - iii. Do the slack pivoting procedure. If all the non-basic slacks are pivoted into the basis, go to v.
   - iv. Do (FSCP).
   - v. Let the resulting assignment be A. Construct a feasible linear precedence on the set of float tasks from A.
2. *Phase II*
   - i. Set C_U to the value min $\{C_{k-1}, C_{\text{Phase I}}\} - 0.1$, where $C_{\text{Phase I}}$ is the final cycle time obtained in Phase I.
   - ii. Using the binary search technique on the cycle time $C \in [C_L, C_U]$, find the minimum cycle time C^* such that (Sub2) has a feasible solution (with cycle time C^*) which requires no greater than w workstations. If such C^* exists, set C_k to C^*, increment k, and go to (step k). If not, go to (*step F*).

(*Step F*) Do post improvement procedure.

C_L is the lower bound on the cycle time which is set to the minimum value between the maximum processing time and the value obtained by dividing the sum of the processing times by the number of workstations. (LBH) solves the problem by iteratively using the algorithms for (Sub1) and (Sub2). Phase I uses the solution methods developed to solve (Sub1). Phase II uses those developed to solve (Sub2). The result of Phase I is used as an input to Phase II and vice versa. Note that the final solution obtained at Phase II is no worse than the solution obtained at Phase I. (LBH) can be viewed as a kind of (somewhat complicated) neighborhood search algorithm, where Phase I generates a neighborhood (in terms of the linear precedence on the set of float tasks) and Phase II finds a local optimal solution in the neighborhood. The following describes some details of (LBH).

7.2. Initial solution

A feasible solution is found by using the modified bin packing method, where the set of workstations corresponds to the set of bins and the set of fixed and float tasks corresponds to the set of items to be packed. Given the current packed items, the items that can be packed in each bin are identified reflecting the constraints on the tasks. Then choose the item to be packed next using the First Fit Method [7].

7.3. Constructing linear precedence on the set of float tasks

At (v) in Phase I, a feasible linear precedence on the set of float tasks is constructed by scanning the assignment A. Since A is a feasible assignment, by considering the set of float tasks assigned to the same workstation from the workstation 1 to w, a feasible linear precedence can be found.

Table 1. Problem characteristics

Problem	w*	m†	n‡	imc-1§	pre¶	imc-2‖
1	37	72	66	19	15	2
2	37	74	63	20	11	2
3	37	74	65	18	9	2
4	35	72	62	23	10	4
5	35	74	61	20	14	3
6	33	74	61	20	11	3
7	33	70	60	22	15	2

*Number of workstations.
†Number of fixed tasks.
‡Number of float tasks.
§Number of incompatibility constraints on the fixed tasks.
¶Number of precedence constraints on the float tasks.
‖Number of incompatibility constraints on the float tasks.

7.4. *Post improvement procedure*

In (*Step F*), the possibility is checked whether a float task can be moved to the other workstation without increasing the current cycle time. If such a movement is possible, a feasible linear precedence on the set of float tasks is constructed using a method similar to the one used at (v) in Phase I, then Phase II is performed subsequently. The procedure is repeated until no improvement upon the current cycle time is possible.

7.5. *Some complications*

As noted in Section 1, there may exist additional constraints on the set of float tasks. There may be partial precedence on the set of float tasks and incompatibility condition on some pairs of float tasks. In these cases, the algorithm proposed in Section 6 should be modified. Here, some possible modifications are presented.

In Phase I, if there exists the above-mentioned constraints, the final assignment can be infeasible. To guarantee feasibility, the algorithm can be modified to consider only the subset of the float tasks for which no additional constraints are present. The float tasks with additional constraints are fixed to the workstations determined in the previous step.

In Phase II, the graph construction procedure should be modified to guarantee feasibility. So, when generating maximal nodes, the additional conditions should be checked and this can be easily implemented.

8. COMPUTATIONAL RESULTS

The proposed algorithm is coded in C language and it is tested using real world data found in a company manufacturing electric washers. Table 1 shows the characteristics of the test problems. Usually, the number of workstations varies between 33 and 37 according to the conditions of the line. The number of fixed tasks is about 72 and that of float tasks is about 63. These and other numbers reveal typical characteristics of the line balancing problem that should be solved in this company.

Table 2 shows computational results. The lower bound on the cycle time (C_L) is calculated simply via the following.

$$C_L = \max\{\max_{i \in M} p_i, \max_{j \in N} a_j, (\sum_{i \in M} p_i + \sum_{j \in N} a_j)/w\}.$$

Table 2. Test results

Problem	Itrn*	C_L†	C‡	Error§
1	7	33	33	0.0
2	8	33	34	3.0
3	10	30	30.7	2.3
4	9	33	35	6.1
5	9	33	33	0.0
6	8	33	39	18.2
7	10	37	41	10.8

*Number of iterations.
†Lower bound on the cycle time.
‡Cycle time obtained.
§Relative error = $(C - C_L)/C_L \times 100$ (%).

Though the lower bound obtained as above may be very poor in some cases, the algorithm could find solutions with small relative errors.

Since the problem is new, it is not possible to compare the performance of the algorithm to other methods. However, the relative errors shown in Table 2 are rather small and it seems that the solutions obtained by the algorithm are near optimal solutions. Previously, the problem was solved manually, which requires several hours. The problem should be solved when a new model is introduced or some changes in the workspace occur. The quality of the solutions obtained by the algorithm seems comparable to the solutions obtained manually, or even improves it. Usually, the computing time requires only a few minutes on a HP715 workstation.

9. CONCLUDING REMARKS

This paper presents a new type of line balancing problem which has practical applications. An efficient heuristic procedure, which is based on some well known network algorithms, is proposed to solve the problem. The heuristic procedure developed is a type of the neighborhood search procedure and an improved solution is constructed by solving two subproblems. One of the subproblems is a generalized bin packing problem for which no algorithms have been presented in the literature. An efficient heuristic based on network theories is proposed for the problem. For the other subproblem, it is shown that it can be solved within polynomial time bound by a shortest path algorithm.

The line balancing problem presented in this paper has many practical applications in its form, but a few refinements can make it possible to apply the model to other situations. For example, parallel workstations consisting of two or more workers can be found in many cases. Further research issues include the extension of the model including such situations and development of optimal algorithms for the models.

A full system which incorporates graphic user interface is developed and successfully installed at the line. Besides providing good solutions when needed, the system can also be used to evaluate several possible alternatives in the process design, which is nearly impossible when the manual procedure is used.

In addition to the increased production rate, a side benefit was obtained using the system. The company operates a system monitoring the quality of the assembled products. The authors were told by the company that the defect rate of the products has decreased after applying the system. The company regarded at least part of this phenomenon can be attributed to the use of a new line balancing system. This side effect was not envisioned by the authors before the system is used and little has been reported on the relationship between the line balancing and product qualities in previous researches. One possible explanation on this phenomenon is that the new system can find a solution which strictly observes the restrictions imposed on the problem while a line designer may find a solution manually that violates some of the constraints for the sake of smaller cycle time and ease of finding the solution. A typical example of the constraints that can be disregarded by a line designer in finding a solution manually involves fuses. If there are two fuses of different capacities to be inserted, it is desirable that the insertion of the fuses be performed by two different workers. If a worker inserts both fuses, he may mistakenly insert the fuses in the wrong place. According to the line manager, this kind of error rarely happens, but it does occur. Also the task of inserting a fuse can be defined as tasks (1) inserting the fuse and (2) inspection and close the lid. It is desirable that these two tasks are performed by two different workers. If a worker performs both tasks, he may mistakenly close the lid without inserting the fuse and the next worker cannot confirm if the fuse has been properly inserted since the lid is already closed. These kind of constraints are not critical for the assembly, but are desirable to observe. But finding a good solution manually, reflecting all these constraints, is not an easy task for a line designer and the line designer may disregard some of the constraints, which can result in quality problems.

REFERENCES

1. Jackson, J. R., Computing procedure for a line balancing problem. *Management Science*, 1956, **2**, 261–271.
2. Baybars, I., A survey of exact algorithms for the simple assembly line balancing problem. *Management Science*, 1986, **32**, 909–932.

3. Talbot, F. B., James, H. P. and Gehrlein, W. V., A comparative evaluation of heuristic line balancing techniques. *Management Science*, 1986, **32,** 430–454.
4. Johnson, R. V., Optimally balancing large assembly lines with 'FABLE'. *Management Science*, 1988, **34,** 240–253.
5. Hoffman, T. R., Eureka: a hybrid system for assembly line balancing. *Management Science*, 1992, **38,** 39–47.
6. Johnson, R. V., A branch and bound algorithm for assembly line balancing problems with formulation irregularities. *Management Science*, 1983, **29,** 1309–1324.
7. Garey, M. R. and Johnson, D. S., *Computers and Intractability*. W. H. Freeman, San Francisco, CA, 1976.
8. Friesen, D. K. and Langston, R. L., Variable sized bin packing. *SIAM Journal on Computing*, 1986, **12,** 60–70.
9. Murty, K. G., *Linear Complementarity, Linear and Non-Linear Programming*. Heldermann, Berlin, 1988.
10. Murty, K. G., *Linear Programming*. Wiley, New York, 1983.
11. Bondy, J. R. and Murty, U. S. R., *Graph Theory with Applications*. Elsevier, Amsterdam, 1976.