JOURNAL OF OPERATIONS MANAGEMENT
Special Issue on Recent Topics in Manufacturing
Vol. 10, No. 4, October 1991

Assembly Line Design with Incompatible Task Assignments

RAM RACHAMADUGU*

EXECUTIVE SUMMARY

Assembly lines are widely used for the mass production of consumer goods and components in large volume production systems. Design of these lines warrants taking into consideration not only cycle time and precedence constraints, but also other restrictions. An important recurring restriction is that some pairs of tasks cannot be assigned to the same station due to factors such as safety, physical demands placed on workers, quality, and technological considerations. We investigate this problem and a current industry practice used for solving this problem. Our investigation of this problem yields three findings. First, we identify a new class of heuristic procedures which dynamically updates task priorities. Our investigations show that this class of heuristics yields better results. Second, extremely greedy procedures such as knapsack heuristics (Hoffmann procedure) continue to perform better than competing heuristics for industrial grade problems even when additional restrictions on task assignments are present. However, when task restrictions are severe, this is no longer true. Lastly, we present an enumerative procedure to search for optimal solutions in the presence of restrictions. Our study shows that while most *simple* assembly line balancing problems can be solved optimally, presence of additional restrictions such as task assignments makes them inherently more difficult. We provide insights into this aspect.

INTRODUCTION

Assembly lines are an efficient means for the mass production of goods and services. In the past four decades, significant research effort has been focused on developing heuristic and optimal methods. Most heuristic methods used priority ranking schemes and probabilistic search techniques, while optimal methods employed enumerative techniques. Many of these techniques focused on designing lines subject to precedence and cycle time restrictions. However, in practice, we come across many other restrictions which complicate the efficient design of lines. One of the reasons cited for infrequent use of the assembly line research by practitioners is that the procedures do not adequately take into account real world needs (Chase (1974)). Some of these practical considerations were discussed in earlier papers by Gehrlein and Patterson (1978); Schoefield (1979); Johnson (1981); Gunther, Johnson and Peterson (1981); and Bartholdi (1991). For a detailed classification of the past research, as well as discussion of restrictions, see the survey article by Ghosh and Gagnon (1990).

We address the issue of task restrictions—i.e., forcing the assignment of tasks to the same station or forbidding the assignment of some task pairs to the same station. These limitations

Manuscript received September 24, 1990; accepted October 21, 1991, after two revisions.

*The University of Toledo, Toledo, OH 43606-3390

were cited by practitioners and academicians in earlier studies (Gehrlein and Patterson (1978), Schoefield (1979), and Farber, Luss and Yu (1987)). Schoefield (1979) discusses these constraints in the context of assembling automotive components. Farber, Luss and Yu (1987) discuss the same in the context of electronic industry. We show in this paper how these practical considerations can be incorporated into heuristic procedures. More importantly, we identify a new class of heuristic procedures. Our computational experiments, along with earlier research results, indicate that the new class of heuristic procedures identified by us for the assembly line balancing problems are competitive and provide improved solutions over their classical counterparts in an average sense. This class of procedures aid practitioners by providing additional alternate solutions to assembly line problems.

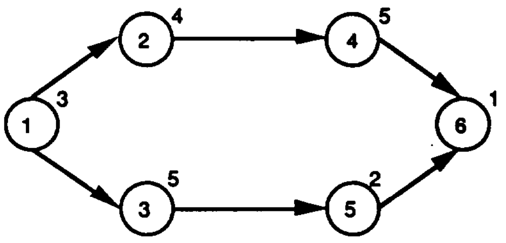
We have developed an enumerative procedure which can be used to solve line balancing problems optimally when task incompatibilities exist. Computational studies with our enumerative procedure show that, while most *simple* line balancing problems can be solved optimally in a reasonable amount of time, they become more difficult to solve with additional restrictions. Our computational studies provide insight into this phenomenon. However, it is gratifying to note that composite heuristic procedures provide optimal or near optimal solutions in most cases.

Our paper is organized as follows: in the next section, we provide a brief description of the problem and also discuss the issue of forced assignment. Forced assignment implies that two or more tasks are forced to be assigned to the same station due to considerations such as common tool requirements. Then we describe knapsack-based procedures used to solve the problem when task incompatibilities are present. Earlier work by Hoffmann (1963); Gehrlein and Patterson (1975, 1978); and Talbot, Patterson and Gehrlein (1986) established the usefulness of knapsack-based procedures for simple assembly line balancing problems. Next we describe a current industry practice for handling task incompatibilities. We then devise a new class of heuristic procedures for assembly line balancing problems. These are called *dynamic priority update procedures* (*DYPUP*). We discuss the merits of this class of procedures over earlier heuristics. Also, we propose a modification of the industry practice for handling task incompatibilities. The proposed modification falls into the DYPUP class. We provide computational results for the heuristic procedures and enumerative procedure that we developed. Finally, we discuss conclusions and future research directions.

PROBLEM DESCRIPTION

Consider the assembly of a product which consists of completing a set of N tasks. In designing paced assembly lines, tasks are assigned to work stations along the line. It is assumed that the tasks cannot be split—i.e., a task is completed at a station where it is begun. Further, precedence restrictions exist among the tasks. This is shown using a precedence graph in Figure 1. A task cannot be performed at a station unless all its predecessors are assigned to the same and/or earlier stations. Also, based on market demands and/or production requirements, a cycle time is specified. Time taken to perform the tasks assigned to a station cannot exceed the cycle time. Also, incompatibilities may exist among tasks. A pair of tasks is considered incompatible if both the tasks cannot be assigned to the same station. These restrictions arise due to practical considerations, such as tool magazine capacity limitations, safety requirements and quality considerations (Farber et al. (1987)). Such tasks are assigned to different stations. In some instances, certain pairs of tasks must be assigned to the same station. Our aim is to develop procedures which require as few stations as possible while satisfying all the above requirements.

FIGURE 1
ALB EXAMPLE PROBLEM WITH TASK INCOMPATIBILITIES



Key: Cycle time = 9

Tasks 1 and 2 are incompatible

Tasks 3 and 5 are incompatible.

First, consider the forced assignment requirements. In some situations, some pairs of tasks need to be assigned to the same station (Gehrlein and Patterson (1975), and Schoefield (1979)). This can easily be handled by collapsing the precedence network. We create a new precedence network in which a single node (or task) represents the set of tasks which needs to be performed at a station (Buxey (1974)). Tasks to be assigned to the same station need not necessarily be adjacent on the precedence network. However, such restriction may necessitate opening parallel stations if the time to perform aggregated tasks exceeds the cycle time. Any heuristic or optimal line balancing method can be used on the revised network.

In the next four sections, we describe procedures which take into account incompatibilities among tasks for balancing assembly lines.

KNAPSACK APPROACH

Most heuristic procedures for assembly lines, with a few notable exceptions which we will address shortly, are bin-packing-based heuristic procedures. These procedures are similar to each other in the sense that tasks are added to a station one-at-a-time until no additional tasks can be assigned to the station without violating one of the constraints. In this section, we describe an alternate approach to the problem which is based on successively solving knapsack problems. Prior to our discussions, we use the concept of *maximal feasible assignment*. This concept was implicitly used by Hoffmann (1963) and explicitly defined by Magazine and Wee (1981). This was also spelled out as a packing principle by Nourie and Venta (1987, 1991).

Definition 1. An assignment of tasks to a station is a *maximal feasible assignment* if no other task can be added to the current assignment without violating the cycle time, precedence and any other constraints.

The procedure we have developed is as follows: from the set of unassigned tasks, choose a subset with the maximum total processing time subject to (i) cycle time constraint, (ii) precedence constraints, and (iii) any other constraints, such as incompatibility constraints. This subset is assigned to the current station and then a new station is opened. The process is repeated

TABLE 1
NOTATION

Notation

N :	Number of tasks
c :	Cycle time
t_i :	Task time for task i, i = 1,2...N
t_{max} :	max t_i, i=1,2...N
IC(i) :	Set of tasks with which task i is incompatible
L :	A large integer constant
D :	A large integer constant used in Hoffmann's procedure
K_i :	Code Number used in Hoffmann's procedure
w_i :	Weight for task i as determined in the FLY procedure
PF_i :	Priority rating for task i in the FLY procedure
S(i) :	Set of all tasks which are successors to task i
IR :	Incompatibility ratio
UAS:	Set of unassigned tasks
ES:	Set of tasks eligible for assignment to the current workstations

Entries in the P matrix represent precedence and incompatibility restrictions among tasks

$$P(i,j): \begin{cases} \left.\begin{array}{ll} 1 & \text{if task is an immediate successor to task i} \\ 0 & \text{otherwise.} \end{array}\right\} i<j \\ 0 \qquad\qquad i=j \\ \left.\begin{array}{ll} -1 & \text{if i and j are incompatible} \\ 0 & \text{otherwise.} \end{array}\right\} j<i \end{cases}$$

P(i): Row vector corresponding to task i in the precedence-incompatibility matrix.

until all tasks have been assigned. Clearly, we solve the following mathematical programming problem at each station

$$\text{Max} \quad \sum_{i \epsilon UAS} t_i x_i$$

$$\text{s.t.} \quad \sum_{i \epsilon UAS} t_i x_i \leq c \qquad (1)$$

$$x_j \leq x_i \qquad j \epsilon S(i) \qquad (2)$$

$$x_i + x_j \leq 1 \qquad j \epsilon \{IC(i) \cap UAS\} \qquad (3)$$

$$x \epsilon \{0,1\}$$

where UAS represents the set of as yet unassigned tasks prior to opening a new station. (1) ensures cycle time feasibility. (2) ensures precedence feasibility. (3) satisfies incompatibility restrictions. If (2) and (3) are deleted, the problem becomes a special case of the knapsack problem in which the reward for including a task equals its processing time. Thus, solving the above problem at a station results in a maximal feasible assignment which has the highest work content. Hence, it is reasonable to expect that this type of procedure would pack the stations as ''densely'' as possible and thus reduce the number of stations. This heuristic argument forms the basis for a class of heuristic procedures which we call knapsack-based heuristics.

Clearly, (P) could be solved using integer programming codes. However, the special structure of the problem (special reward function and precedence constraints represented by (2)) lends itself for the development of specialized procedures. This fact was originally exploited by Hoffmann (1963). However, his procedure was not intended to take into account incompatibility constraints represented by (3). The reader is referred to the original paper by Hoffmann (1963) for further details. Labach (1990) addressed the issue of how such procedures perform in more complicated environments, such as mixed model assembly lines.

At this juncture, we comment on the solution generated by solving (P) for assembly line balancing problems.

Remark 1. Workload assigned to the first work station by using (P) is not less than the workload assigned to the first work station by any heuristic or optimal procedure for the assembly line problem.

Proof follows immediately from the definition and is omitted here for the sake of brevity.

The Hoffmann procedure solves a special case of (P), yielding solutions in which total work assigned to the first station is not less than the work assigned to the first station in any heuristic or optimal procedure. If the reverse network is solved, then the statement is true for the last work station. It is reasonable to expect that the solution procedures for assembly lines based on successively solving (P) for each work station would perform well vis-à-vis other heuristic procedures which construct maximal feasible assignments using only one task at a time. This was earlier verified by Talbot et al. (1986) in their extensive computational studies. These factors motivated us to develop a heuristic procedure which repeatedly solves (P) at each work station. We modified the Hoffmann procedure to account for additional restrictions imposed by incompatibility constraints. An interesting feature of our modification is that we use the precedence matrix to represent incompatibility restrictions as well (details are provided in the Appendix). We call this procedure the *modified knapsack procedure* (*MODKNAP*).

INDUSTRY PRACTICE

In this section, we describe a current industry practice for balancing assembly lines when incompatibilities exist among some pairs of tasks. Farber, Luss, and Yu (1987) discuss the problem of assigning tasks to stations in electronic assembly. They cite situations which forbid assignment of certain pairs of tasks to the same station. For example, tooling capacity restrictions may force certain pairs of tasks to be assigned to different work stations. Also, some components may be so similar to each other that quality considerations forbid the insertion of these components on the board at the same station. They developed a heuristic procedure for designing the assembly lines, which we call the Farber, Luss and Yu (FLY) procedure.

In the FLY procedure, tasks are assigned weights (wi) and priorities (PF_i) so that (i) a task will not appear in the list before its predecessors have appeared, (ii) subject to satisfying (i), a task with larger number of incompatibilities appears before a task with lesser number of incompatibilities, and (iii) subject to satisfying (i) and (ii), a task with higher ranked positional weight (Helgeson and Birnie (1961)) appears earlier than a task with lower ranked positional weight. Thus the *weights* assigned to the tasks are as follows:

$$w_i = L^*|IC(i)| + t_i \tag{4}$$

The *priority* assigned to a task equals the sum of the weights of all its successors plus its own weight:

$$PF_i = \sum_{j \in S(i)} w_j \tag{5}$$

Tasks are arranged in the decreasing order of PF_i values. Each task is assigned to the earliest station at which it can be fitted without violating the precedence, cycle time and incompatibility restriction. If a task cannot be fitted into an existing station, then a new station is opened. As an illustration, consider the problem shown in Figure 1. Weights and priorities assigned by the FLY procedure to various tasks are shown in Table 2. Tasks 1 and 3 are assigned to station 1. Since task 2 does not fit into station 1, a new station is opened. Tasks 2 and 5 are assigned to station 2. Tasks 4 and 6 are assigned to station 3.Some comments about the FLY procedure are noteworthy here. FLY procedure falls into the broad class of heuristic procedures known as rank and assign methods (Baybars (1985)). Also, the reader will notice that, in the absence of incompatibilities, the FLY procedure reduces to the well-known ranked positional weight (RPW) method (Helgeson and Birnie (1961)).

TABLE 2
PRIORITIES FOR VARIOUS TASKS UNDER MPP SCHEME
PRIOR TO ASSIGNING TASKS AT STATION 1

Task	1	2	3	4	5	6
w_i	13	15	14	12	5	1
PF_i	60	28	20	13	6	1

L = 10

DYNAMIC PRIORITY UPDATE PROCEDURES (DPP)

Earlier researchers used priority schemes such as the ranked positional weight, task duration, number of successors, etc. (Talbot et al. (1986)) for assigning priorities to tasks. These schemes assigned priorities to tasks at the start of the heuristic procedure and the priorities remained invariant during the assignment of tasks to stations. However, heuristic procedures used by earlier researchers differed from each other in other ways. Rank and Assign (Baybars (1986)), Generalized First Fit Decreasing (Wee and Magazine (1981)), Immediate Update First Fit Decreasing (Hackman, Magazine and Wee (1990)) differed from each other in determining when a task became *eligible* for assignment. Further, in all these classes of implementation, an eligible task with lower priority was not assigned unless an eligible task with higher priority was already assigned. Thus, it is conceivable that during the task assignment process, a station may have a nonmaximal feasible assignment before a new work station was opened. Heuristic procedures can also be implemented in a slightly different way. A new work station need not be opened unless a maximal feasible assignment is obtained at the current work station. The solution can be built in stages, one work station at a time. Thus, an eligible task with higher priority can remain unassigned even when a task with lower priority has already been assigned to the current station. However, an earlier station never needs to be examined for subsequent task assignments since it has a maximal feasible assignment.

From the above discussion, it is clear that all prior implementations of heuristic priority rules used static priority weights for the jobs. Next, we describe a new *class* of heuristic procedures—dynamic priority update procedures (DYPUP). DYPUP differs from the four classes of heuristic procedures proposed by earlier researchers for the assembly line balancing problems. DYPUP updates the priorities for all unassigned tasks whenever a new station is opened and/or a task is assigned to the current station. The logic is explained in Figure 2. Though such priority updates do not alter the relative ranking of tasks in most heuristic procedures discussed in the literature (which are subject only to precedence and cycle time feasibility constraints—the reversed ranked positional weight method is a notable exception), presence of additional restrictions can change priorities and the relative ranking of tasks.

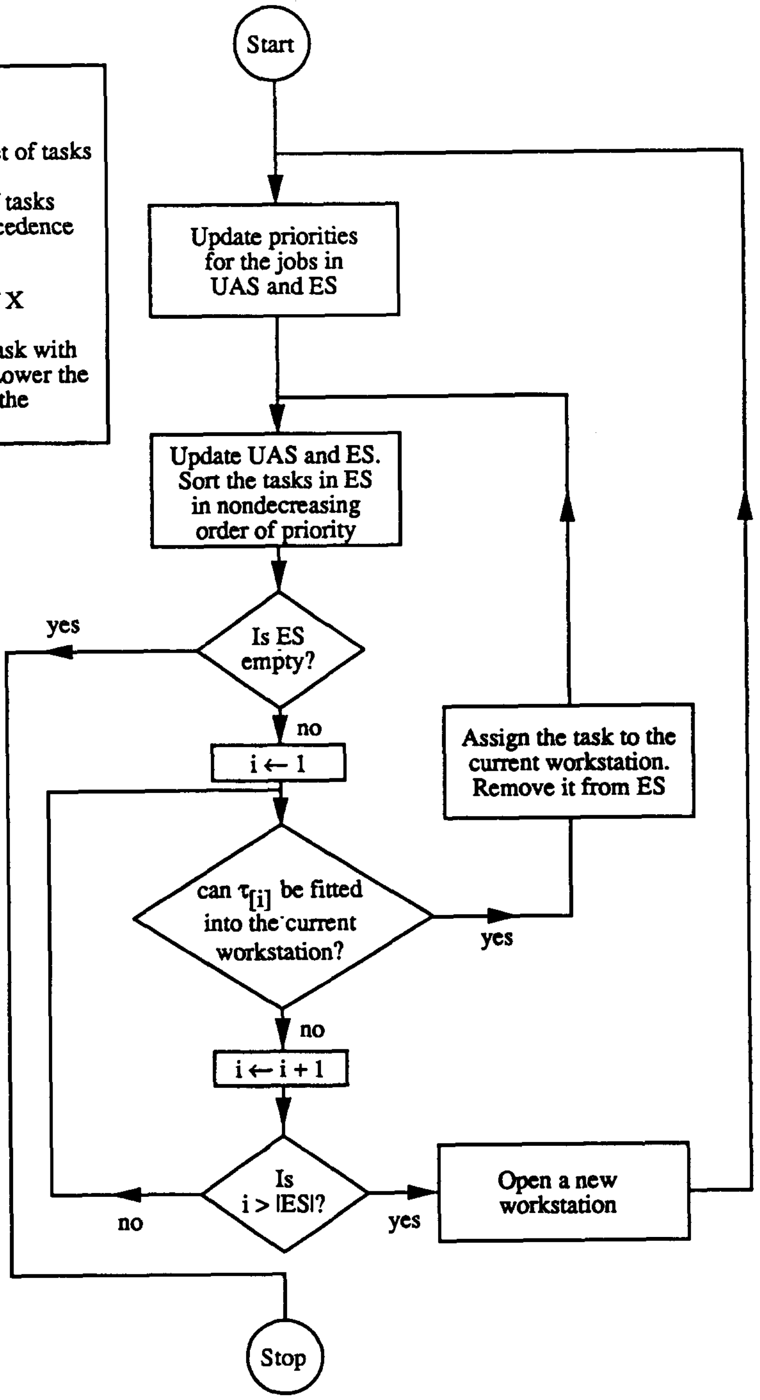
We now describe a heuristic procedure for the task incompatibility problem which falls into DYPUP category. In the procedure suggested by Faber, Luss and Yu (1987), priority assigned to a task remains the same all through the analysis. Even after assigning some tasks to stations, priorities are not reassessed. However, whenever a new station is opened, we can reassess the priorities of unassigned tasks by considering incompatibilities only among the unassigned tasks. Since the incompatibility between the tasks in the unassigned set and the tasks assigned to earlier work stations is irrelevant for future task assignments, we anticipate that the dynamic updating of priorities will likely lead to a better balance. Thus, in the procedure developed by us, priorities are assigned to the jobs using the same scheme as in the FLY procedure. However, these priorities are reassessed for all unassigned tasks when a new station is opened. For further discussion, a modified version of the industry practice is called *modified priority procedure* (*MPP*).

As an illustration, once again consider the problem in Figure 1. Priorities for various tasks under the MPP scheme prior to assigning tasks at station 1 are already shown in Table 2. Tasks 1 and 3 are assigned to work station 1. Before assigning tasks to station 2, we reassess priorities for all unassigned tasks. These values are shown in Table 3. Note that task 5 now has higher priority than task 2. Tasks 5 and 2 are assigned to station 2. Tasks 4 and 6 remain unassigned. However, their PFi values are unchanged. Tasks 4 and 6 are assigned to station 3.

FIGURE 2
DYNAMIC PRIORITY UPDATE PROCEDURE

NOTATION

UAS:	Unassigned set of tasks
ES:	Eligible set of tasks (based on precedence feasibility)
$\lvert X \rvert$:	Cardinality of X
$\tau_{[i]}$:	Index of the task with ith priority. Lower the index, higher the priority.



Start
Update priorities for the jobs in UAS and ES
Update UAS and ES. Sort the tasks in ES in nondecreasing order of priority
Is ES empty?
yes
no
$i \leftarrow 1$
can $\tau_{[i]}$ be fitted into the current workstation?
yes
Assign the task to the current workstation. Remove it from ES
no
$i \leftarrow i + 1$
Is $i > \lvert ES \rvert$?
no
yes
Open a new workstation
Stop

TABLE 3
PRIORITIES FOR ALL UNASSIGNED TASKS

Task	2	4	5	6
w_i	4	5	2	1
PF_i	10	6	3	1

COMPUTATIONAL STUDIES

Our computational study consisted of three phases. The first phase studied the relative merits of the three heuristic procedures (FLY, MPP and MODKNAP) using a set of eight problems from the industry. The second phase studied the relative performance of the heuristics using a set of challenging problems from the literature. These problems were modified to control for varying degrees of incompatibility among the tasks. The third phase describes the development and performance of an enumerative procedure for the problem. We provide the computational results and analysis in the next three subsections.

Tests on Industry Data Sets

We tested FLY, MPP and MODKNAP procedures on eight data sets from the electronics industry. A particular feature of these problems was that task times were not randomly distributed, but fell into few discrete categories. This is due to the fact that most operations involved insertion of standardized and somewhat similar components on printed circuit boards. Brief details are shown in Table 4. Number of tasks in the data sets varied from 12 to 101. Order strength varied from 0 to 0.166. The degree of incompatibility among tasks is shown by the number of incompatible pairs of tasks. It varied from 1 to 179.

TABLE 4
RESULTS FOR INDUSTRY DATA SETS

Problem #	1	2	3	4	5	6	7	8
Number of tasks	18	25	12	14	23	137	56	101
Pairs of incompatible tasks	1	21	12	8	4	1	2	179
Order strength	.013	.026	0	0	.134	.166	.0045	.090
Tasks/station*	3.6	3.57	4	3.5	3.83	15.22	5.6	10.1

*The ratio is based on the minimum number of stations found.

For all eight data sets, FLY and MPP resulted in the same number of stations. This is not surprising since the extent of incompatibility among tasks is not severe. We will study this aspect in more detail in the next subsection. In the case of data sets 1-6, MODKNAP needed one station less than FLY and MPP. Also, these solutions were optimal since the number of stations were equal to the lower bound obtained by relaxing the indivisibility and precedence relationships among the tasks. In case of data sets 7-8, all procedures yielded the same number of stations. These results show that the modified version of Hoffmann procedure (MODKNAP) performs better than the competing procedures on our industry data sets.

Comparative Tests on Literature Problems

We further tested the procedures on a set of challenging problems from the literature. This was done for the following reasons: first, the data sets from the industry were characterized by "lumpiness" of processing times. There is no a priori reason to believe that this is true of all industrial problems. Second, it is not clear how the severity of incompatibility among tasks will affect the relative performance of the procedures. Suppose that all tasks are incompatible with each other. Clearly, the number of stations required will equal the number of tasks. The result would be independent of the procedure used, task times, and cycle times. At the other extreme, if there are no incompatibilities present at all, conclusions drawn by Talbot et al. (1986) regarding relative performance of the procedures hold valid. Thus, the need exists for controlled experimentation regarding "severity" of incompatibility and its impact on the relative performance of the procedures under investigation. Also, results from the industry data sets could not discriminate between FLY and MPP.

Prior comparative computational studies on assembly line design did not include task incompatibilities. Hence no prior measures exist to define the degree of incompatibility among tasks. We devised a measure called the Incompatibility Ratio (IR) to measure the severity of incompatibility among the tasks. For any problem, IR is determined as follows:

$$\text{Incompatibility Ratio (IR)} = \frac{\text{Number of incompatible task pairs}}{\frac{N(N-1)}{2}}$$

A task pair is considered incompatible if both tasks cannot be assigned to the same station. The denominator in the above expression is the total number of distinct task pairs in a problem. IR $=0.1$ implies that, in an average sense, a task is incompatible with 10% of the tasks. When IR $=0$, there are no incompatibilities among tasks. In that case, both FLY and MPP yield identical solutions. When IR $=1$, all tasks are incompatible with each other and all procedures (heuristic and optimal) will result in as many stations as the number of tasks in the problem. IR ratios for the industrial problems shown in Table 4 are .0065, .0700, .1818, .0879, .0158, .0001, .0013, and .0355.

There are no prior benchmark problems from the literature in which incompatibilities were studied. However, a few comparative studies exist (Baybars (1986), Mastor (1970), Johnson (1981), and Talbot et al. (1986)). The problems identified and complied by Talbot et al. (1986) have been widely studied. Recently, Hoffmann (1990) identified a set of 28 problems as challenging. These are Sawyer's problem, Tonge's problem, Arcus-83 and Arcus-111 for certain cycle times. The interested reader is directed to Hoffmann (1990) for further details. We used these 28 problems as a computational benchmark. We generated problems for our experiment by randomly imposing incompatibility restrictions among tasks at the following IR levels (0, 0.025, 0.05, 0.1 and 0.2). Thus, the three procedures were tested on 140 problems (28 problems x5 IR levels). We chose these IR values since the industry problems had values in this range.

Table 5 provides a summary of the results. The numbers in the table are a relative measure of heuristic performance. Each problem was solved using all three heuristic procedures. Performance of each procedure was compared with the minimum value among the three procedures. Averages of these values are shown in Table 5. Earlier, Talbot et al. (1986) used this measure in their study. We used this as a metric since optimal solutions are difficult to find for this set of problems (we elaborate on this issue in the next subsection). Further, from the earlier study by Talbot et al. (1986), it appears that the relative ranking of procedures based on this measure is a good surrogate for the ranking based on deviation from the optimal solution.

TABLE 5
AVERAGE PERCENTAGE ABOVE MINIMUM HEURISTIC VALUE FOR PROBLEMS FROM THE LITERATURE

Incompatibility Ratio (IR)	0	0.025	0.05	0.10	0.20
Industry Practice (FLY)	1.99	2.76	2.21	2.38	2.27
Modified Priority Procedure (MPP)	1.99	2.12	1.98	2.07	1.05
Modified Knapsack procedure (MODKNAP)	0.29	0.80	1.62	1.57	6.04

Results corresponding to IR = 0 reaffirm conclusions from earlier studies that the Hoffmann procedure performs better than competing procedures for the simple assembly line balancing problem. Though MODKNAP performs better than the industry practice at low levels of IR, its performance deteriorates at large values of IR. At IR = 0.2, MPP performed better than MODKNAP in 18 cases whereas MODKNAP could provide better solutions in three cases. These results contradict the superior performance of knapsack-based heuristic procedures in earlier research studies. However, those studies investigated only simple assembly line balancing problems in which cycle time and precedence feasibility were the only constraints.

Intuitively, MODKNAP is appealing in the sense that it assigns to each successive station the best possible subset of tasks from as yet unassigned tasks. Hence we would expect it to perform better than other heuristic procedures which load the stations one task-at-a-time. On further analysis, it appears that in its attempt to load each station as much as possible, MODKNAP delays the assignment of extremely incompatible tasks to later stations. Since these unscheduled tasks are incompatible among themselves, the procedure results in opening up many stations even when the stations are under loaded. Further, the reader may note that the incompatibility restriction tends to be more "severe" than the precedence and cycle time constraints. If a task is incompatible with a large number of tasks, its assignment to a particular station will preclude assigning the set or subset of incompatible tasks to the same station. Since MODKNAP disregards this aspect, its performance seems to deteriorate when severe incompatibility restrictions are present. However, we should note that, in the range of incompatibilities that we had come across from the industry data, MODKNAP performs better than the other two competing procedures. It should also be noted that MODKNAP is computationally more expensive than FLY and MPP procedures. However, this is not a critical issue since we are dealing with a design problem that does not warrant real-time computations, and in any case, solution times are less than a second on average for a problem on an IBM 3090-600E system (with IBM FORTRANVS compiler invoked at optimization level 3).

We note that MPP seems to improve upon FLY in an average sense. Also, results of some early studies lead us to believe that dynamic priority update procedures (DYPUP) may yield better solutions than static priority procedures. Consider the reverse positional weight method discussed by Magazine and Wee (1981). Clearly, its dynamic priority update version is first-fit-decreasing method. Magazine and Wee's computational results show that first-fit-decreasing method performs better than the reverse positional weight method (Tables 2 and 3, Wee and Magazine (1981)). Thus, the class of dynamic priority update procedures constitute a new and useful set of heuristics for assembly line problems.

Development of Enumerative Procedure

As part of our study, we developed an implicit enumeration procedure for solving the assembly line problems optimally with additional restrictions such as incompatibilities. The procedure builds one station at a time. It generates lexicographically ordered maximal feasible assignments for each successive station. The partial solution is pruned if the lower bound on the partial solution equals or exceeds the current upper bound. When a solution is pruned, the next maximal feasible assignment is generated for the last station in the partial sequence. Else, the process is repeated for the next station. When all tasks are assigned, the upper bound is updated, if necessary. This is done whenever a better feasible solution is found. Also, the search is terminated if the upper bound equals the theoretical minimum number of stations. The process backtracks to the previous station whenever the current station is fathomed either by the lower bound on the partial solution or no further maximal feasible assignments exist. The procedure is repeated until there are no more maximal feasible assignments to be generated for the first station or the optimality of the current upper bound has been verified by comparing it with the theoretical minimum. The procedure is analogous to OPTPack (Nourie and Venta (1991)), but takes into account incompatibility restrictions and differs in data structure details. The initial upper bound was set at the minimum value obtained from the three heuristics.

The procedure was coded in FORTRAN77 and run on an IBM 3090-600E system at the University of Michigan, Ann Arbor, using the IBM FortranVS compiler. The code optimizer was invoked at level 3. Since the challenging problem set was expected to be difficult, CPU time was limited to 180 seconds for each problem. Our computational results are shown in Table 6. IR = 0 represents the challenging set of problems identified by Hoffmann (1990). IR = 0.2 represents the same set of problems with incompatibility restrictions imposed on tasks. IR = 0.2 implies that, on average each task is incompatible with 20% of the tasks.

First, we note that our procedure solved all except four challenging problems (IR = 0). We also provide solutions for some unreported cases. Note that the optimum was never more than one station above the theoretical minimum. Also, the gap between the initial solution and optimum was never more than two stations. Hence, the optimum solution is straddled by a tight lower bound and a good initial upper bound, with a gap of no more than two stations.

TABLE 6
COMPUTATIONAL RESULTS FOR THE CHALLENGING SET

			IR = 0			IR = 0.2		
Problem	Cycle Time	Theoretical Minimum	Best Initial Solution	Optimum Stations	CPU Time (seconds)	Best Initial Solution	Optimum Stations	CPU Time (seconds)
Sawyer's 30 Element	25	13	14	14	0.099	15	14	21.327
Sawyer's 30 Element	27	12	13	13	0.065	14	13	20.362
Sawyer's 30 Element	30	11	12	12	13.216	12	12	0.414
Sawyer's 30 Element	33	10	11	11	6.668	11	11	0.194
Sawyer's 30 Element	36	9	10	10	0.236	11	10	0.221
Sawyer's 30 Element	41	8	8	8	**	10	9	0.105
Sawyer's 30 Element	47	7	8	7	4.087	9	8	0.061
Sawyer's 30 Element	54	6	7	7	0.238	8	7	0.253
Sawyer's 30 Element	65	5	6	5	0.030	7	7	0.141
Sawyer's 30 Element	81	4	4	4	**	8	8	0.138
Sawyer's 30 Element	108	3	3	3	**	8	8	0.060
Tonge's 70 Element	156	23	24	23	24.295	26	≤ 25,*	> 180
Tonge's 70 Element	160	22	23	23	2.835	26	23	39.005
Tonge's 70 Element	168	21	23	22	85.988	24	22	114.189
Tonge's 70 Element	185	19	20	20	6.090	22	*	> 180
Tonge's 70 Element	207	17	18	18	80.909	19	*	> 180
Tonge's 70 Element	220	16	17	17	49.993	18	*	> 180
Tonge's 70 Element	251	14	15	14	19.335	17	15	153.194
Tonge's 70 Element	293	12	13	13	31.192	17	*	> 180
Arcus 83 Element	3691	21	22	*	>180	23	*	> 180
Arcus 83 Element	3786	20	21	*	>180	24	*	> 180
Arcus 111 Element+	5689	27	28	*	>180	29	*	> 180
Arcus 111 Element+	5785	26	27	26	0.058	29	*	> 180
Arcus 111 Element+	6016	25	27	25	14.462	28	*	> 180
Arcus 111 Element+	6267	24	25	25	0.061	28	*	> 180
Arcus 111 Element+	6837	22	23	23	0.061	25	*	> 180
Arcus 111 Element+	7520	20	21	*	>180	22	*	> 180
Arcus 111 Element+	7916	19	20	20	0.061	22	*	> 180

Note: + Computations for these problems were carried out with the task time for 84th task set at 2889, as done in earlier studies by Talbot, et al. [1986].
* Optimum could not be found within 180 seconds CPU time. If the procedure found better solution than the initial solution within the CPU run time, we reported the number of stations in the solution.
** Initial solution is optimum since it equals the theoretical minimum.

When incompatibilities are present, problems appear to be more difficult to solve. While we were able to solve all but four problems in the simple case (IR = 0), we were unable to solve fourteen of the challenging problems when incompatibilities are present (IR = 0.2). Of the fourteen problems we were able to solve, in nine cases computing time was larger with task incompatibilities. It is also interesting to note that, while the gap between the optimum and the theoretical minimum was no more than one station in simple cases (IR = 0), the gap tends to increase with cycle times for the incompatibility case. This can be seen in the case of Sawyer's problem. Also, our studies with the sixty-four literature problems (Talbot et al. (1986)) showed the occurrence of similar phenomenon, in cases where optimum solutions could be found. When cycle times are large, task incompatibility becomes more restrictive than the task times in assigning the tasks to stations. The gap between the optimum and the theoretical minimum contributes to the diminished effectiveness of the theoretical minimum as a powerful pruning device. Also, the need to ensure compatibility among tasks assigned to a station adds to the computational burden. These factors provide some insights as to why the problems with task incompatibilities or additional restrictions are more difficult to solve optimally when compared to the simple case.

CONCLUSIONS

Our paper addressed the assembly line design problem which takes into consideration additional practical restrictions such as task incompatibilities. Our investigations resulted in a new class of heuristic procedures in which priorities assigned to the tasks are dynamically updated. This class of heuristic procedures is different from earlier classes of heuristic procedures. Our computational tests based on the challenging data sets from the literature and analysis of a current industry practice show that dynamically adjusting the priorities yields better results. This is also substantiated by the synthesis of earlier research studies.

We also investigated the knapsack approach to the assembly line design with task restrictions. Earlier studies on lines with only cycle time and precedence restrictions showed that these procedures (Hoffmann (1963), Talbot et al. (1986) and Labach (1990)) performed well. We developed a new procedure (solving knapsack problems with item incompatibilities and rewards being proportionate to item sizes) which takes into account task restrictions. Our computational experiments show that the knapsack-based procedure developed by us performed better than the industry practice for the sample of industrial grade problems. However, when the incompatibility restrictions were set high in the challenging set from the literature, its performance deteriorated. Hence, it is necessary to validate procedures for the assembly line design by testing them on relevant and appropriate industrial grade problems.

Our computational studies also provide some insights into the nature of assembly line problems. When the line design is subject to only cycle time and precedence constraints, optimal solutions for the problems studied in the literature are no more than one station above the optimum. This is also true for the challenging problem set proposed by Hoffmann (1990). However, our computational studies show that when there are additional practical considerations such as task incompatibilities, the gap between the theoretical minimum and the optimum can be more than one station. Hence, even if an enumerative procedure and/or a heuristic procedure finds an optimal solution, its verification can take a large amount of computational time. Saltzman and Baybars (1987) also comment on this aspect. Our findings prompt two additional avenues of research. It needs to be investigated if currently available effective enumerative methods such as FABLE (Johnson (1988)), OPTPack (Nourie and Venta (1991)), and Hoffmann (1988) can perform well in the presence of additional practical considerations such as task incompatibilities. The other avenue is to use composite objective functions for the knapsack approach, as suggested by a referee. The advantage of the latter approach is that, by varying the values of the coefficients, it may not only give improved solutions, but also generate alternate solutions with the same number of stations. Since practitioners tend to use many nonquantitative criteria in their choices, they may find this approach attractive and useful.

ACKNOWLEDGMENT

I thank the referees for their constructive and critical comments on earlier versions of the paper. I also acknowledge the support of professional peers who promptly provided me with data sets, papers and necessary clarifications.

REFERENCES

Bartholdi, III, J.J. "An Interactive Program to Balance Assembly Lines." Presented at the TIMS/ORSA Joint National Meeting, Las Vegas, Nevada, May 1990.

Baybars, I. "A Survey of Inexact Algorithms for the Simple Assembly Line Balancing Problem." Working Paper, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA, 1985.

Baybars, I. "A Survey of Exact Algorithms for the Simple Assembly Line Balancing Problem." *Management Science,* vol. 32, no. 8, August 1986, 909-932.

Buxey, G.M. "Assembly Line Balancing with Multiple Stations." *Management Science,* vol. 20, no. 6, June 1974, 1010-1021.

Chase, R.B. "Survey of Paced Assembly Lines." *Industrial Engineering,* vol. 6, February 1974, 14-18.

Farber, M., H. Luss, and C.S. Yu. "Assembly Line Design Tools: Line Balancing and Line Layout." *In Proceedings of the Second International Conference on Production Systems,* Paris, France, 1987.

Gehrlein, W.V., and J.H. Patterson. "Sequencing for Assembly Lines with Integer Task Times." *Management Science,* vol. 21, no. 9, September 1975, 1064-70.

Gehrlein, W.V., and J.H. Patterson. "Balancing Single Model Assembly Lines: Comments on a Paper by E. M. Dar-El (Mansoor)." *AIIE Transactions,* March 1978, 109-12.

Ghosh, S., and R.J. Gagnon. "A Comprehensive Literature Review and Analysis of the Design, Balancing and Scheduling of Assembly Systems." *International Journal of Production Research,* vol. 27, no. 4, 1989, 637-670.

Gunther, R.E., G.D. Johnson, and R.S. Peterson. "Currently Practiced Formulations for the Assembly Line Balance Problem." *Journal of Operations Management,* vol. 3, no. 4, 1981, 209-221.

Hackman, S.T., M.J. Magazine, and T.S. Wee. "Fast, Effective Algorithms for Simple Assembly Line Balancing Problems." *Operations Research,* vol. 37, no. 6, November-December 1988, 916-924.

Helgeson, W.P., and D.P. Birnie. "Assembly Line Balancing Using Ranked Positional Weight Method." *Journal of Industrial Engineering,* vol. 12, no. 6, 1961, 394-98.

Hoffmann, T.R. "Assembly Line Balancing with a Precedence Matrix." *Management Science,* vol. 9, no. 4, April 1963, 551-62.

Hoffmann, T.R. "Assembly Line Balancing by Implicit Complete Enumeration." Working Paper, Management Sciences Department, School of Management, University of Minnesota, Minneapolis, Minnesota, 1988.

Hoffmann, T.R. "Assembly Line Balancing: A Set of Challenging Problems." *International Journal of Production Research,* vol. 28, no. 10, 1990, 1807-1815.

Johnson, R.V. "Assembly Line Balancing Algorithms: Computational Comparisons." *International Journal of Production Research,* vol. 19, no. 3, 1981, 277-287.

Johnson, R.V. "A Branch and Bound Algorithm for Assembly Line Balancing Problems with Formulation Irregularities." *Management Science,* vol. 29, no. 11, November 1983, 1309-1324.

Johnson, R.V. "Optimally Balancing Large Assembly Lines with FABLE." *Management Science,* vol. 34, no. 2, February 1988, 240-253.

Labach, E.J. "A Comparative Evaluation of Heuristic Sequencing and Line Balancing Techniques for Mixed Model Assembly Lines." Working Paper, School of Management, Boston University, Boston, MA, 1990.

Magazine, M.J., and T.S. Wee. "Fast Algorithms for the Assembly Line Balancing Problems." Working Paper No. 149, Department of Management Sciences, University of Waterloo, Waterloo, Ontario, Canada, 1981.

Magazine, M.J., and T.S. Wee. "An Efficient Branch and Bound Algorithm for the Assembly Line Balancing Problem." Working Paper No. 150, Department of Management Sciences, University of Waterloo, Waterloo, Ontario, Canada, 1981.

Mastor, A.A. "An Experimental Investigation and Comparative Evaluation of Production Line Balancing Techniques." *Management Science,* vol. 16, 1970, 728-745.

Nourie, F.J., and E.R. Venta. "The Packing Heuristic for Assembly Line Balancing Problems." *Journal of Applied Business Research,* vol. 3, no. 3, Fall 1987, 122-128.

Nourie, F.J., and E.R. Venta. "Finding Optimal Line Balances with OPTPACK." *Operations Research Letters,* 1991.

Saltzman, M.J., and I. Baybars. "A Two-Process Implicit Enumeration Algorithm for the Simple Assembly Line Balancing Problem." *European Journal of the Operational Research,* vol. 32, 1987, 118-129.

Schoefield, N.A. "Assembly Line Balancing and the Application of Computer Techniques." *Computers and Industrial Engineering*, vol. 3, 1979, 157-173.

Talbot, F.B., J.H. Patterson, and W.V. Gehrlein. "A Comparative Evaluation of Heuristic Line Balancing Techniques." *Management Science*, vol. 32, no. 4, April 1986, 430-54.

APPENDIX

We describe here how modifications can be made to Hoffmann's procedure to take into account incompatibility restrictions. The reader is assumed to be familiar with the procedure described by Hoffmann (1963). We describe below only the changes to be made in the procedure devised by Hoffmann. Step numbers refer to the steps enumerated by Hoffmann (1963).

Define for each task i, two new row vectors, M(i) and Q(i), each with n elements, as follows:

$$M(i) = (0, 0\ldots.. 0, p(i + 1, i), p(i + 2, i)\ldots.. p(n, i)).$$

$$Q(i) = (p(i, 1), p(i, 2)\ldots.. p(i, i - 1), 0, 0\ldots\ldots..).$$

Note that M(i) represents the set of tasks which are incompatible with task i and which have indices greater than i. Similarly Q(i) represents the set of tasks which are incompatible with task i and which have indices smaller than i. (Q(l) and M(n) are zero row vectors.) M(i) and Q(i) are constructed whenever necessary by looking up the precedence-incompatibility matrix.

Additional necessary changes to Hoffmann's code in order to take incompatibilities into account are as follows.

Step 5 : Suppose that the element selected is i. Subtract from the Code Number the row corresponding to i and also M(i). Use this result as a new Code Number. Go to Step 6.

Step 10: Replace the first Code Number with the last Code Number corresponding to the previous result. Add to the Code Number $\sum_{i \in s} (M(i) + Q(i))$ where s represents the set of tasks assigned at the previous station.

Changes made in Step 5 ensure that if a task is selected for assignment at a work station, then none of the tasks which are incompatible with it are selected for assignment at that work station.

Changes made in Step 10 ensure that whenever a new work station is opened, all tasks whose predecessors have already been assigned will be considered for assignment at the new work station.

Next, we describe how the modified Hoffmann procedure works by illustrating the procedure on the example problem shown in Figure 1 in the paper.

Calculations for the Example Problem (Figure 1 in the Text) with Task Incompatibilities:

Task

Task	1	2	3	4	5	6	Element Selected	Element Time	Station Time Remaining Before Selection	Station Time Remaining After Selection	Elements Contained in Combination After Selection	
1	D	1	1	0	0	0						
2	-1	D	0	1	0	0						
3	0	0	D	0	1	0						
4	0	0	0	D	0	1						
5	0	0	-1	0	D	1						
6	0	0	0	0	0	D						
Combination Column ↓												↓Comments
K_1	$\underset{\uparrow}{0}$	1	1	1	1	2	1	3	9	6	1	
	D +0	1 -1	1 0	0 0	0 0	0 0						Subtract (P(1)+Q(1))
K_2	-D	1	$\underset{\uparrow}{0}$	1	1	2	3	5	6	1	1,3	
	0 0	0 0	D 0	0 0	1 -1	0 0						Subtract (P(3)+Q(3))
*K_3	-D	1	-D	1	1	2						
K_2	-D	1	0	1	1	2						

* indicates the latest combination generated at each station in step 8.

——— in the combination column indicates the generation of a maximal feasible combination.

═══ in the combination column indicated the completion of assignment at a station.

K_1	0	1	1	1	1	2						
K_3	-D	1	-D	1	1	2						
	0	-1	0	0	0	0						Add (M(1)+Q(1))
	0	0	0	0	-1	0						Add (M(3)+Q(3))
K_3	-D	$\underset{\uparrow}{0}$	-D	1	0	2	2	4	9	5	2	
	-1	D	0	1	0	0						Subtract
	0	0	0	0	0	0						(P(2)+Q(2))
K_4	-D+1	-D	-D	$\underset{\uparrow}{0}$	0	2	4	5	5	0	2,4	
	0	0	0	D	0	1						Subtract
	0	0	0	0	0	0						(P(4)+Q(4))
*K_5	-D+1	-D	-D	-D	$\underset{\uparrow}{0}$	1	5	2	0		2	
K_4	-D+1	-D	-D	0	$\underset{\uparrow}{0}$	2	5	2	5	3	2,5	
	0	0	-1	0	D	1						Subtract
	0	0	0	0	0	0						(P(5)+Q(5))
K_5	-D+1	-D	-D+1	0	-D	1			3			
K_4	-D+1	-D	-D	0	0	2						
K_3	-D	0	-D	1	$\underset{\uparrow}{0}$	2	5	2	9	7	5	
	0	0	-1	0	D	1						Subtract
	0	0	0	0	0	0						(P(5)+Q(5))
K_4	-D	0	-D+1	1	-D	1			7			
K_3	-D	0	-D	1	0	2						

K_5	-D+1	-D	-D	-D	0	1						
	-1	0	0	0	0	0						Add (M(2)+Q(2))
	0	0	0	0	0	0						Add (M(4)+Q(4))
K_5	-D	-D	-D	-D	0 ↑	1	5	2	9	7	5	
	0 0	0 0	-1 0	0 0	D 1	1 0						Subtract (P(5)+Q(5))
K_6	-D	-D	-D+1	-D	-D	0 ↑	6	1	7	6	5,6	
	0 0	0 0	0 0	0 0	0 0	D 0						Subtract (P(6)+Q(6))
*K_7	-D	-D	-D+1	-D	-D	-D			6		5,6	