A Methodology To Solve Single-model, Stochastic Assembly Line Balancing Problem And Its Extensions

by

Erdal Erel

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in
Industrial Engineering and Operations Research

APPROVED:

Subhash C. Sarin, Chairman

Aseem S. Chandawarkar

Wolter J. Fabrycky

Marilyn S. Jones

Bradley O. Skarpness

May 1987
Blacksburg, Virginia

A Methodology To Solve Single-model, Stochastic Assembly Line Balancing Problem And Its Extensions

by

Erdal Erel

Subhash C. Sarin, Chairman

Industrial Engineering and Operations Research

(ABSTRACT)

A methodology for the solution of single-model, stochastic assembly line balancing problem is developed for the objective of minimizing the total labor cost (dictated by the number of stations on the line) and expected incompletion cost arising from tasks not completed within the prespecified cycle time.

The proposed procedure is an approximation procedure that divides the problem into subproblems. For each subproblem, an approximate solution is obtained using the dynamic programming procedure developed for the problem. This procedure is incorporated with a special bounding strategy to overcome the rapidly increasing storage and computational requirements as the size of the problem increases. These approximate solutions are further improved by a branch-and-bound type of procedure called the improvement procedure. This procedure uses approximate costs, instead of lower bounds, to fathom the nodes of the enumeration tree constructed; thus, it is not, in true sense of the word, the branch-and-bound technique. Consequently, the procedure is not guaranteed to result in the optimal solution; however, it is shown to generate solutions within $(1+\varepsilon)$ of the optimal solution. The improvement procedure either improves the approximate solutions obtained using the dynamic programming procedure or determines that they are quite close to the optimal ones. The improved solutions of the subproblems are then appended to each other to produce the solution of the original problem.

Some dominance properties that contribute to the effectiveness of the improvement procedure and help in reducing the size of the enumeration tree are developed. Some sequencing and scheduling problems related to the node evaluation scheme of the improvement procedure are also investigated. A single-machine sequencing procedure is developed for the objective of minimizing the expected incompletion cost with tasks having a common due date and stochastic processing times. This procedure is extended to construct a schedule on M parallel machines. In these procedures, incompletion costs of the tasks are independent of their expected performance times; it can be interpreted as relaxing the precedence relations among the tasks. Solution procedures are also developed for the above sequencing and scheduling problems for the case in which the incompletion costs of the tasks are proportional to their expected performance times. Computational results and analyses made indicate that these procedures result in almost optimal solutions.

Acknowledgements

I would like to express my gratitude and respect to Dr. Subhash C. Sarin for his invaluable guidance, assistance and infinite patience in directing this research. It was a great pleasure to work with him, in particular, to take part in those intense brainstorming sessions.

I am also deeply indebted to the members of my committe, Dr. Aseem S. Chandawarkar, Dr. Wolter J. Fabrycky, Dr. Marilyn S. Jones and Dr. Bradley O. Skarpness, for their conscientious consideration of the dissertation and helpful suggestions during the course of this work. I would also like to thank my colleagues, too numerous to list, who have assisted me with their help, encouragement and guidance.

Finally, I would like to dedicate this work to my mother, Suat Erel, and to the memory of my father, Celal Erel. They instilled in me a belief in the value of education, and their faith, love, and belief in me have been the most important influences in my life.

Table of Contents

List of Illustrations

List of Tables

LIST OF SYMBOLS

A_i	set of tasks that cannot get started due to the incompletion of task i
A'_i	set of tasks following task i on the precedence diagram
APP_i	approximate cost of node i in the enumeration tree of the improvement procedure
b_i	station to which task i is assigned
B_i	set of tasks preceding task i in the station
c_k	number of tasks with label k
C	cycle time
CIC_i	cumulative incompletion cost of task i
CIN_{iM}	expected incompletion cost of the M-station solution of the relaxed problem corresponding to node i in the enumeration tree of the improvement procedure
CN_i	total expected cost associated with assignment of tasks to node i in the enumeration tree of the improvement procedure
CRX_i	expected incompletion cost of the solution of the relaxed problem corresponding to node i of the improvement procedure
$CW^j_{i,k}$	set of complete tasks in the kth case of the enumeration tree constructed for T^j_i
D	demand rate
f_i	flowtime of task i
G_i	set of tasks in node i and all its parent nodes in the enumeration tree of the improvement procedure
h_i	label of task i
H_i	set of tasks following task i in the station
IC_i	incompletion cost of task i
$IW^j_{i,k}$	set of incomplete tasks in the kth case of the enumeration tree constructed for T^j_i
K	total number of stations on the line
L	labor rate
n	number of nodes in the enumeration tree of the improvement procedure

N	number of tasks of the line balancing problem
N_{ir}	number of tasks in the relaxed problem corresponding to node i in the enumeration tree of the improvement procedure
N_{sp}	maximum number of tasks allowed in a subproblem
$NSTA_{DP}$	number of stations in the dynamic programming procedure solution
o_i	occurrence probability of the node with task i being completed in the probability enumeration tree
P_i	set of tasks preceding task i on the precedence diagram
q_i	number of tasks assigned to station i
Q_i	set of tasks immediately preceding task i on the precedence diagram
R_i	set of tasks in node i in the enumeration tree of the improvement procedure
s_n	state variable at stage n of the dynamic programming procedure
S_k	station time of station k
SB_i^j	overcounted incompletion costs corresponding to T_i^j
ST_k	set of tasks in station k
t_i	task performance time of task i
T_i	set of tasks preceding task i in the station that do not precede task i on the precedence diagram
T_i^j	jth starting event of task i in which the tasks in TS_i^j and task i can be started and the tasks in TN_i^j cannot be started
TCN_i	total expected cost associated with assignment of tasks to node i and all its parent nodes in the enumeration tree of the improvement procedure
UB_{cur}	current upper bound in the improvement procedure
$w_{i,k}^j$	occurrence probability of the kth case of the enumeration tree constructed for T_i^j
W	set of all the tasks in the line balancing problem
x_n	decision variable at stage n of the dynamic programming procedure
X_n	set of all possible decision variables at stage n of the dynamic programming procedure

Y	number of zeros in the precedence matrix
α	bound on the incompletion probability of the tasks in a decision variable of the dynamic programming procedure
β_i	probability that task i is started to be processed and not completed
β_i^i	probability that T_i^i occurs and task i is incomplete while the tasks in TS_i^i are completed
γ_i^i	probability that T_i^i occurs
Γ_i^i	probability that task i is incomplete while the tasks in TS_i^i are completed
μ_i	expected performance time of task i
σ_i	standard deviation of the performance time of task i
ζ_i	parent node of node i in the enumeration tree of the improvement procedure

1.0 Introduction

1.1 Types Of Production And Characteristics Of Flow Production

A production system can be considered to consist basically of an input of raw materials and/or components and a conversion unit which changes the state of the input to form an output of end products which are delivered to a customer [88]. There are several methods of classifying production systems for descriptive purposes. The simpliest classification is based on the concepts of continuous and intermittent processes. Continuous production involves the production of an item by using the conversion units for 24 hours a day and 365 days per year. In contrast, intermittent production involves basically the manufacturing of an item, with absolutely no repetition.

A classification which is perhaps more useful for our purposes relies on the division of production systems into four broad and overlapping groups, namely continuous, mass and flow, batch, and job production.

- Continuous production involves a one-product system with a very large and continuous demand for the product. Petroleum refining is an example of this group.

- Mass and flow production involves a small variety of products with a large demand. Mass production requires a large demand while flow production requires a large and continuous demand. Mass production does not need to be flow production, but flow production is invariably also mass production.

- Batch production involves a large variety of products but with small demand for each one.

- Job production involves basically manufacturing a single, unique item.

A production or flow line may be defined as an arrangement of manufacturing facilities in such a way that they perform successive operations on the product(s) manufactured. A production line may be designed to produce or assemble a single product (in which case production is more or less continuous), or a family of similar products which require the same production facilities but may have different sequences of operations. Principles of flow-line production can be summarized as follows [107]:

1. Principle of work flow. The work, material or products should flow smoothly and regularly through the production process or facilities.

2. Use or provision of interchangeable parts. Assembly lines depend on the availability of interchangeable parts, and flow lines used in 'machining' items are required to produce items to sufficient standards of accuracy to ensure their interchangeability.

3. Principle of minimum distance moved. To ensure continuity of flow, and maximum utilization of available space, it is essential that the flow pattern should be both logical and efficient.

4. Division of operation. Although not strictly a principle, the division, rationalization or specialization of operations is an important feature of flow line production.

1.2 Definitions And Terminology

An assembly line can be considered as a production sequence where parts are assembled together to form an end product with the assembly operations being carried out at work stations situated along the line. The basic characteristic of an assembly line is the movement of the workpiece from one work station to the next.

We present the definitions of several terms related with assembly lines below. A classification of assembly lines will be given in the next section.

Station is a location on the line at which work is performed on the product either by adding parts or by completing the assembly operations. Stations can be 'closed' or 'open'. It is undesirable, or impossible, for operators from adjacent stations to violate the boundaries of a closed station. Open station boundaries can be crossed, so there is flexibility in the times available for completing tasks allocated to open stations, but it is required that no interference occurs between adjacent operators [22]. **Station time** is the actual amount of work, in time units, assigned to a specific station on the assembly line. The station time of station k and the total number of stations are denoted by S_k and K, respectively.

Work element or **task** is a rational division of the total work content in an assembly process. Elements should be small enough in content to combine well with other elements. On the other hand, they should not be divided too finely, because that permits nonproductive work to creep into the line. Work elements, or tasks, should be defined to give minimum total work content [44]. The total number of work elements of a problem is denoted by N.

Work element time or **task time** or **task performance time** is the duration to perform the task. The task time of task i is denoted t_i. Most of the techniques developed to solve assembly line balancing problems assume this value to be a deterministic constant. On the other hand, the majority of the assembly lines in industry consist of worker performed tasks with variable performance times. Task time variabilities become quite large relative to their means when the tasks are complex and demand high level of skill and concentration. For such cases, task performance times are assumed to be distributed according to a probability distribution function, where μ_i and σ_i^2 denote the mean and the variance of the performance time of task i, respectively.

Cycle time is the amount of time a unit being worked on is available to an operator. It is represented by C and is assumed to be constant for all operators for a given conveyor speed. A lower bound on cycle time is the maximum of the work element times. A tighter lower bound is the maximum station time. An upper bound on C is imposed by the demand rate; the reciprocal of demand rate constitutes an upper bound. That is,

$$\underset{i=1,..,N}{\text{Max}}\ t_i \le \underset{j=1,..,K}{\text{Max}}\ S_j \le C \le \frac{1}{D}$$

where D is the demand rate.

Idle time is the difference between the cycle time and station time. It is conventional to take the sum of all station idle times as a measure of the efficiency of the design of a line; the summation is called **total idle time**. A related measure of efficiency is **balance delay** which is the ratio of the total idle time and the total time spent by the product in moving from the beginning to the end of the line [54]. It can be expressed as follows:

$$\text{Balance Delay} = \frac{100\left[K\,C - \sum_{i=1}^{N} t_i\right]}{K\,C}$$

In a perfectly balanced line, balance delay is zero. Another related measure of efficiency is **smoothness index**. It can be expressed as follows:

$$\text{Smoothness Index} = \sqrt{\sum_{j=1}^{K} (S_{max} - S_j)^2}$$

A perfect balance would have a smoothness index of zero.

Precedence diagram is a graphical description of any ordering in which work elements (tasks) must be performed in achieving the total assembly of the product. An example of a precedence diagram for an 11-task problem is depicted in Figure 1. The numbers inside the circles identify the tasks. The diagram is drawn so that the assembly progresses from left to right. The tasks in a column are mutually independent, and can be permuted among themselves without violating order restrictions. **Precedence matrix** is an upper-triangular matrix which has an entry of one for the ith row and jth column if task j follows task i in the precedence diagram. Otherwise, the entry is zero. The precedence matrix of the example depicted in Figure 1 is shown in Figure 2.

The precedence structure of an assembly task can be characterized by the **Flexibility ratio** , or **F-ratio.** F-ratio is a measure of the number of feasible sequences that could be generated from an N-element assembly task. It is expressed as follows:

$$\text{F-ratio} = \frac{2Y}{N\,(N - 1)}$$

where Y is the number of zeros in the precedence matrix. F-ratio ranges from zero for precedence diagrams ordered serially to one for diagrams having no precedence relationships. The F-ratio of the precedence diagram depicted in Figure 1 is 0.418. Note that the F-ratio of a precedence diagram

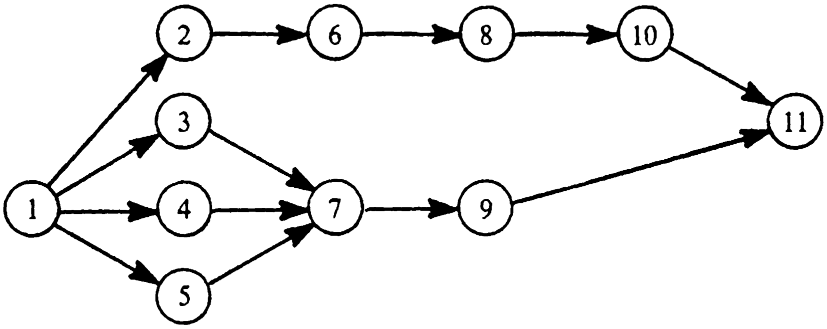


Figure 1. Precedence diagram of a 11-task problem

Tasks

		1	2	3	4	5	6	7	8	9	10	11
	1		1	1	1	1	1	1	1	1	1	1
	2			0	0	0	1	0	1	0	1	1
	3				0	0	0	1	0	1	0	1
	4					0	0	1	0	1	0	1
Tasks	5						0	1	0	1	0	1
	6							0	1	0	1	1
	7								0	1	0	1
	8									0	1	1
	9										0	1
	10											1

Figure 2. Precedence matrix of the precedence diagram depicted in Figure 1

is proportional to the number of feasible sequences that could be generated. **West ratio** is the average number of tasks assigned to stations. It is expressed as follows:

$$\text{West Ratio} = \frac{N}{K}$$

Order strength is a measure of the volume of distinct orderings that are permitted by the specified precedence relations. It is defined as the ratio of the number of ordering relations to the possible number of ordering relations and expressed as follows:

$$\text{Order Strength} = \frac{\text{Number of precedence relations}}{\frac{N(N-1)}{2}}$$

The order strength of the precedence diagram depicted in Figure 1 is 0.582. As order strength decreases, the number of feasible sequences of the tasks increases and consequently it requires more computing effort for the solution techniques developed.

Paralleling of tasks is allowing tasks to be performed at more than one station. **Paralleling of stations** is allowing equivalent work stations at certain points along the line. Paralleling concepts are used to modify a line design in several ways; balance efficiency can be improved, the limitation imposed by the longest task time on C can be relaxed, etc.

Blocking of a station occurs when a station must hold a serviced unit because due to lack of space to deposit the unit ready for the next station. **Starving of a station** occurs when a station (except the first) has finished serving a unit, ejected it, but finds no unit queueing in its storage bank and will, therefore, have to wait for units being processed by the preceding stations.

Zoning constraints are the additional constraints on the assembly line balancing problem, such as some tasks may be impossible to be performed at the same station, or in a different context some tasks may only be performed at the same station.

The **launching discipline** is the rule for feeding items into the line. There are two launching disciplines. The first is 'fixed rate launching', where the launching period is equal to the cycle time. The second discipline is 'variable rate launching', where the launching period is the station time of the first station of the last product launched. The latter can be used in multi-model or mixed-model lines.

The **incompletion cost** is the cost of completing the task off the line. Incompletion cost of task i is denoted by IC_i. It is a parameter of stochastic assembly line balancing problems. Since the task times are assumed to be distributed according to a probability distribution function, there is a probability that the tasks in a station could not be completed within the cycle time. Such tasks are called incomplete tasks and are completed off the line. Note that decreasing the cycle time or the number of stations of an assembly line increases the incompletions. A task gets incomplete due to two reasons: 1) the task is not completed within the cycle time, or 2) the task is a follower of another incomplete task on the precedence diagram. Incompletion costs depend on the way incompletions are handled. If incompletions are completed off the line, then the cost includes the labor cost of completing the task off the line, the cost of machinery used and the other related overhead costs. On the other hand, if incompletions are scrapped, then the cost includes the value of the item at that point and the cost associated with the starved stations on the line due to the incompletion. When a task gets incomplete, a set of tasks may also get incomplete since some tasks cannot get started to be processed. This set of tasks depends on the precedence diagram and the allocations of tasks to stations. In other words, the incompletion of a task can cause some other tasks to get incomplete. Let task i be the incomplete task. The summation of the incompletion cost of task i and those of the tasks that cannot get started to be processed is called the **cumulative incompletion cost** of task i and denoted by CIC_i.

1.3 Statement Of The Assembly Line Balancing Problem

Assembly line balancing problems can be classified into four categories: Single-model deterministic, single-model stochastic, multi/mixed-model deterministic and multi/mixed-model stochastic. The single-model deterministic version of the problem applies to single-model assembly lines where the task performance times are known constants. This is the simpliest form of the assembly line balancing problem. The single-model stochastic category relaxes the deterministic task performance time assumption and introduces the concept of task time variability. This version models the manual assembly lines more realistically where task performance times are seldom constants. The multi/mixed-model deterministic version of the problem introduces the concept of producing more than one item on a single line. Multi-model lines are involved in the production of two or more similar types of items produced separately in batches, and mixed-model lines produce two or more similar items simultaneously. The multi/mixed-model stochastic category is the most complex version of the problem to analyze.

Another classification of assembly lines depends on the way the items are moved through the line. In this classification there are basically two distinct types; nonmechanical and moving belt lines. Operators on nonmechanical lines are normally free of any mechanical pacing effect; product remains stationary at each station, and after the completion of the station's tasks, it is either transferred to the next station or the next station's operators replace the current one. Moving belt lines are basically paced lines characterized by a conveyor belt. Moving belt lines have two subcategories, namely those in which items are, or are not, removable from the belt.

In this dissertation the single-model, stochastic assembly line balancing problem is addressed. Since the issues involved and the objective criteria considered in the deterministic and stochastic versions of the problem are quite different from each other, the statements of the two versions of the problem will be presented separately.

1.3.1 Statement Of The Deterministic Assembly Line Balancing Problem

The deterministic assembly line balancing problem can be stated as follows: Given a finite set of tasks, each having a fixed performance time, and a set of precedence relations which specify the permissible orderings of the tasks, tasks must be assigned to an ordered sequence of stations such that the precedence relations are satisfied and some measure of performance is optimized. The sum of the performance times of the tasks assigned to a station should be less than or equal to the cycle time. The most commonly used performance measure is the minimization of the number of stations on the line. The assumptions of the deterministic assembly line balancing problem are as follows:

1. Task times are known constants.

2. The precedence diagram for the problem is known with certainty.

3. No splitting of the tasks among stations is permitted.

4. Each station is manned by one worker who is paid the same wage regardless of his assignment.

5. Each task can only be started if all its predecessors have been completed.

The techniques developed for the solution of the deterministic assembly line balancing problem have used several performance measures. The most commonly used performance measure is to minimize the number of stations. Minimizing the total idle time, minimizing the cycle time for a given number of stations, and minimizing the balance delay are some other performance measures used by the techniques reported in the literature. Most of the performance measures can be reduced to each other easily; the objective function for minimizing the total idle time can be expressed as follows:

$$\text{Minimize total idle time} = \text{Min } Z = K\,C - \sum_{i=1}^{N} t_i$$

The objective as stated above can be reduced to one of two alternative forms as follows:

1. Min Z = K, given C, or,
2. Min Z = C, given K.

The reduction is due to the fact that $\sum_{i=1}^{N} t_i$ is a constant and K or C are the only variables to be minimized. In most cases, cycle time is predetermined when management sets the production rate, or an upper bound is imposed by production planning requirements. Thus, the problem is reduced to finding the minimum number of stations. In particular, the problem is reduced to minimization of the number of stations or number of operators subject to the constraints:

1. All tasks have to be performed,

2. No task can be assigned more than once,

3. The work content in any station cannot exceed the cycle time,

4. If task x precedes task y on the precedence diagram, then y cannot be allocated to a station that precedes the one to which x is assigned.

Although the problem is easy to formulate, the enumeration of the feasible task sequences to find the minimum number of stations requires an enormous effort. It consistently defies the development of efficient algorithms for obtaining optimal solutions. The problem has a finite but extremely large number of feasible solutions; this immense computational complexity and the problem's inherent integer restrictions result in enormous computational difficulties. Without the precedence constraints, there are N! different sequences of N tasks. For as few as twenty tasks, N! is approximately 2.4×10^{18} , and the enumeration of this many sequences is beyond the capacity of any computer. On the other hand, precedence and cycle time constraints reduce this figure drastically. As Ignall [44] reports "if there are r precedence relations among N tasks (r arrows on the precedence diagram), then there are roughly $\frac{N!}{2^r}$ distinct sequences". For twenty tasks with thirty arrows on the precedence diagram, the figure reduces to 2.3×10^9 which is still a very large value to enumerate and evaluate.

1.3.2 Statement Of The Stochastic Assembly Line Balancing Problem

The stochastic assembly line balancing problem can be stated as follows: Given a finite set of tasks, each having a performance time distributed according to a probability distribution, and a set of precedence relations which specify the permissible orderings of the tasks, the problem is to assign the tasks to an ordered sequence of stations such that the precedence relations are satisfied and some measure of performance is optimized. The stochastic assembly line balancing problem is obtained by relaxing the first assumption of the deterministic assembly line balancing problem, specifically that the task performance times are random variables instead of constants.

When task performance times are considered to be random variables, objective criteria different from the ones for the deterministic version should be used, since the cost associated with incompletions becomes an important issue. The sum of the performance times of the tasks assigned to a station may exceed the cycle time. The most commonly used performance measures are the minimization of the probability that one or more stations exceed the cycle time or the minimization of the total system cost which consists of the total labor cost (a function of the number of operators employed) plus the total expected incompletion cost arising from tasks not being completed as units move down the line. When the task performance times are stochastic, the incompletion cost becomes an important part of the total system cost. The stochastic assembly line balancing problem addressed in this dissertation has the objective of minimizing the total system cost. The objective function of the model can be expressed as follows:

Min Z = Total Labor Cost + Total Expected Incompletion Cost

The value of Z depends on the number of stations and the allocations of tasks to the stations. The optimal value of Z can be obtained by varying K and the allocations of tasks to these stations, such that

1. All tasks are allocated to stations,

2. No task is allocated more than once,

3. If task x precedes task y on the precedence diagram, then y is not allocated to a station that precedes the one to which x is assigned.

The enumeration and evaluation of the feasible sequences for the stochastic version of the problem is much more complex than that of the deterministic case. First of all, there are more feasible sequences, since the cycle time constraint cannot be applied any more. In addition, the evaluation process is quite time-consuming.

For a given number of stations on the line, the computation of the total labor cost term is straightforward; it is linearly proportional to the number of stations, and the proportionality constant is the labor cost of a station. On the other hand, the computation of the total expected incompletion cost term is much more complex and involves the computations of several other variables. For a given number of stations and allocations of tasks to these stations, the following variables should be determined for each task in order to compute the total expected incompletion cost term: (i) the probability that the task can be started to be processed, (ii) the probability that the task is not completed within the cycle time after it has been started to be processed, and (iii) the cost incurred when the task is not completed within the cycle time. Note that a task can get started to be processed if a certain set of tasks are completed. If a task is not completed within the cycle time, then another task may not get started. The same task may also not get started due to the incompletion of another task. If these incomplete tasks are independent of each other (if they are in different stations), then the incompletion cost of the task that cannot get started is overcounted with some probability. Such overcounting of incompletion costs should be subtracted from the total expected incompletion cost expression to obtain the exact value. The determination and computation of these overcounted costs contribute to the difficulty of computing the total expected incompletion cost term. A detailed discussion of the variables that should be determined in order to compute the terms of the above objective function is given in Chapter 3. The derivation of each variable is also discussed, and a general expression which captures the cost factors of the objective function is developed.

1.4 Purpose Of This Research

The development of the first real example of an assembly line is credited to Henry Ford in 1913. But, for over forty years, only trial and error methods were used for the solution of the assembly line balancing problem. A survey taken in the USA [14] in 1969 indicated that still 42% of the assembly lines were balanced by trial and error methods. This value would be a conservative estimate for other countries. Of the remaining, some 40% used manual applications of a recognized technique which is either very time consuming or is an approximate procedure for a given application. Chase's [17] survey of 95 companies revealed that only 5% of the companies were using published techniques to balance their lines. This suggests that either currently available techniques are inadequate to model the actual conditions of assembly lines, or the practitioners are unfamiliar with the published algorithms.

The purpose of this research is to develop and implement an efficient methodology to solve the single-model, stochastic assembly line balancing problem for the objective of minimizing the total labor cost (which is dictated by the number of stations) and the total expected incompletion cost which is incurred if the assigned work at a station is not completed within the prespecified cycle time. The methodology is based on the approximation procedure which divides the problem into subproblems. For each subproblem, an approximate solution is first obtained and these approximate solutions are further improved. The solutions of the subproblems are combined to produce the solution of the original problem. The procedure does not guarantee attainment of the optimal solution but a detailed experimentation is carried out to show that the procedure is very efficient to use and it generates better solutions than those of the other techniques reported in the literature. In addition, some sequencing and scheduling problems related to the procedure are also investigated.

1.5 Dissertation Outline

This dissertation will be organized as follows: Chapter 2 describes some of the more pertinent literature published in the field. Chapter 3 presents the development of the cost model. Chapter 4 presents the dynamic programming formulation of the problem and an implementation strategy is given for the formulation. Chapter 5 presents the approximation procedure and the other procedures on which the approximation procedure is based. In Chapter 6, the methodology is extended by relaxing some of the assumptions of the problem. Solution procedures to some sequencing and scheduling problems are also presented in Chapter 6. Finally, in Chapter 7 we conclude and discuss areas of future research and extensions.

2.0 Literature Review

The first published analytical statement of the assembly line balancing problem was made by Salveson [85] in 1955. Since 1955 several papers on developing, improving and comparing line balancing methodologies have appeared in the literature. Especially in the last two decades, several assumptions of the earlier formulations have been relaxed, and the techniques developed have been applied to larger and more realistic line balancing problems.

This chapter will survey some of the techniques reported in the literature for the solution of assembly line balancing problem. Since the techniques developed for assembly lines with deterministic task times are quite different than the ones for lines with stochastic task times, we will survey the literature in two sections: deterministic assembly line balancing and stochastic assembly line balancing. The first section is partitioned into two parts: algorithms and heuristic procedures.

2.1 Deterministic Assembly Line Balancing

2.1.1 Algorithms

2.1.1.1 Integer Programming Formulations

The conceptual formulation of the deterministic assembly line balancing problem is as follows:

Minimize Total Idle Time

subject to

1. All tasks have to be performed,
2. No task can be assigned more than once,
3. The work content in any station cannot exceed the cycle time,
4. If task x precedes task y on the precedence diagram, then y cannot be allocated to a station that preccedes the one to which x is assigned.

Let W be the set of tasks to be performed on one unit, and ST_j be the set of tasks contained in the jth. station. Then, the conceptual formulation can be represented as:

$$\text{Min } Z = KC - \sum_{i=1}^{N} t_i$$

subject to

$$\bigcup_{j=1}^{K} ST_j = W$$

$$ST_j \cap ST_i = \emptyset \qquad \text{for } i, j = 1,\ldots,K \text{ and } i \neq j$$

$$S_j = \sum_{k \in ST_j} t_k \leq C \quad \text{for } j = 1,\ldots,K$$

$$\text{If } x \leq y \text{ and } x \in ST_i,\ y \in ST_j, \text{ then } i \leq j$$

Assembly line balancing with linear integer programming was first treated by Salveson [85] for the objective of minimizing total idle time. The constraints of his model are as follows: i) precedence relations, ii) a task is assigned once and only to one station, and iii) the sum of task times at any station is less than or equal to the cycle time. The formulation is identical to the conceptual formulation given above. He suggested two computational procedures depending on different conditions: Lines with relatively few precedence constraints versus many constraints. The first is based on a given cycle time and the objective is to minimize wasted or unused operator time. The second one assumes a fixed number of stations and the number of stations is varied discretely over a range predetermined so as to span the desired rate of output. Unfortunately, the methods are not workable for practical problems of realistic size. The enumeration of all possible stations is enormous and as the line gets bigger, the approach seems to be of academic rather than practical interest. As Kilbridge and Wester [55] state, "Industrial engineers using traditional trial and error techniques can generally arrive at as good a balance with less work".

Bowman [9] presented two separate linear integer programming formulations of the problem. In the first model, the objective function is to make later stations exceedingly costly, pushing the operations as far forward as is physically possible. The constraints of the model are; i) none of the stations are overloaded, ii) each operation is performed, iii) operations are not split between stations, and iv) precedence relations. In his second model, the notion of 'clock time' when an operation is started is utilized. The objective function is to minimize the cycle time. This formulation is superior to the previous one as the number of variables and constraint equations is much less. However, the computations required for a balancing problem of even modest size are still considerable. For a problem with 8 tasks, 7 maximum number of stations and 8 orderings, the first formulation requires 135 constraint equations and 112 variables and the second formulation requires 33 constraint equations and 24 variables. Thus, considering the lines of industry, Bowman's approach to the problem is of more academic than practical value.

White [106] modified Bowman's first formulation by introducing binary variables indicating the assignment of tasks to stations. The new integer programming formulation requires 71 constraint equations and 56 variables for the same problem. Thangavelu and Shetty [98] further extended Bowman's formulation of the problem which requires only 24 constraint equations and 56 variables and presented a solution procedure based on Balas' additive algorithm.

Patterson and Albracht [70] presented an integer programming formulation of the problem which is more efficient than the other formulations in the literature. Precedence relations are used to advantage in eliminating from consideration a significant number of variables. They also developed a search algorithm similar to the procedure of Thangavelu and Shetty [98]. Bowman's [9] 8-task problem requires 29 variables and 23 constraint equations. The search algorithm for a 70-task problem which involves 770 variables took 9.803 seconds of CPU time on an IBM 370/168.

Roberts and Villa [81] presented an integer programming model for a multi-product assembly line balancing problem. The number of stations is the same for all products. However, due to the large number of variables and constraint equations, the formulation is of more theoretical than practical interest. For example, a 2-product problem involving a total of 16 tasks and 16 precedence relations requires 126 variables and 60 constraints.

Pinto, Dannenbring and Khumawala [73] formulated an integer programming model for the assembly line balancing problem with paralleling of the tasks and presented a branch-and-bound type of algorithm for its solution. The effect of paralleling of tasks is to allow a task to be performed at more than one station, thereby reducing the effective task time by the number of times the facility is replicated. They assumed that the task times of the two paralleled tasks are equal to one-half the task time of the original unparalleled task. Another assumption of the model is that paralleled tasks cannot be assigned to the same station. The branch-and-bound type algorithm proceeds by partitioning the set of all combinations into subsets of 'partial' combinations. A partial combination is made up of tasks which can be classified into three mutually exclusive states: 'fixed' paralleled, 'fixed' not paralleled, and undecided. An undecided state is equivalent to not paralleling. A 'full' combination is achieved by fixing all tasks either to be paralleled or not paralleled. Pinto, Dannenbring and Khumawala [75] later formulated the problem which allowed paralleling of stations. The main idea of paralleling stations is that labor cost might be reduced, as the paralleled stations furnish greater flexibility in assigning tasks to stations, and it allows the production rate to be greater than is achievable with conventional models. Their formulation relaxes one of the constraints of the conceptual formulation expressed above, namely, the cycle time constraint is no longer active. Later, the authors [76] extended the formulation to include processing alternatives. Choice of processing alternatives and assignment of tasks to stations are considered simultaneously, because the achievable savings can only be determined after an assignment of tasks to stations has been made. They formulated the problem as an integer linear programming problem and presented a branch-and-bound type of procedure for the solution in which the branches correspond to the designated facility choices. A problem with 50 tasks and only three processing alternatives required 6.8 seconds of CPU time on an IBM 370/158.

Talbot and Patterson [97] developed an algorithm which minimize the station number to which the unique terminal task was assigned. The solution procedure to the formulation makes a systematic enumeration of all possible task assignments. An artifice called network cut eliminates the assignment of tasks to stations where such assignments would not lead to improved balances. The procedure can obtain optimal balances in a reasonable amount of time for lines consisting of 50 or fewer tasks.

2.1.1.2 Shortest-Route Formulations

The general shortest-route formulation of the assembly line balancing problem can be described as constructing a directed network which have nodes representing the possible assignments of the tasks. The generation of the nodes should be mutually exclusive and totally exhaustive, so that all possible combinations are generated. The path from source to sink node which requires the least number of nodes becomes the optimal solution.

Klein [56] formulated the assembly line balancing problem as an assignment problem or as a shortest-route problem. A network with directed arcs which represent a possible assignment of tasks to a station is constructed. Every path from source to sink becomes a possible line design. The method is not suitable for large problems, since it constructs the set of all feasible orderings of the tasks and a shortest route problem is solved for each feasible order. As he stated, "a small problem with nine tasks takes about an hour of manual computing time".

Gutjahr and Nemhauser [38] proposed an algorithm to solve the problem based on finding a shortest route in a finite directed network. The nodes represent a collection or subset of tasks that can be performed in some order without prior completion of any task not in the subset. The directed arc (ij) is defined if and only if U_i is a subset of U_j and $t(U_j) - t(U_i) \leq C$, where U_i and $t(U_i)$ represent the set of tasks at node i and the sum of their performance times, respectively. The major difficulty with this algorithm is that the number of nodes generated increases exponentially with problem size. The method is a considerable improvement over the shortest-route approach of Klein [56]; since the number of arcs is much smaller. Another advantage of the approach is that to do a sensitivity analysis on C is quite simple: If a new cycle time $C_1 < C$ is tried, then finding the nodes only with arcs entering with time greater than C_1 is necessary. The test problems solved on an IBM 7090 gave the following execution times: A 9-task problem in less than 1 second, a 14-task problem in 3 seconds, and a 17-task problem in 180 seconds.

Roberts and Villa [81] extended the algorithm given by Gutjahr and Nemhauser [38] to multi-product line balancing problems. But, the storage requirements of the model are very demanding, and the computational requirements for practical sized problems are staggering.

2.1.1.3 Dynamic Programming Formulations

The general dynamic programming formulation of the assembly line balancing problem has stages representing stations and states representing the set of tasks unassigned. For each state, the feasible station assignments are generated at each stage. The ordering of the tasks which requires the least number of stations is the optimal solution to the problem.

Jackson [45] is one of the first pioneers to solve the assembly line balancing problem. He proposed an algorithm which finds an optimal solution if carried to completion. Although not formulated in today's usual dynamic programming terminology, it is well tailored to fit the deterministic assembly line balancing problem. The method constructs all feasible first-station assignments; then for each such first-station assignment, it constructs all feasible second-station assignments; for each first-second combination, it constructs all feasible third-station assignments, and so forth. He presented some refinements to his formulation which are tests to eliminate the need of further considering some stations as candidates to lead to optimality. Jackson's algoritm is conceptually flawless, but in application the computational work for lines of practical sizes is frequently impractical.

Johnson [47] proposed a technique to solve the balancing problem with some formulation irregularities. The technique is mainly identical to Jackson's [45] approach: A tree of solutions is formed, where each feasible station is represented by an arc of the tree. Formulation irregularities, such as planned imbalance of station times and assigning tasks to particular types of stations are incorporated into generation of the tree of solutions. The approach seems to have large storage requirements for problems of realistic sizes. Several 20-task problems were solved on a DEC VAX/780 system to see the performance of the approach to various irregularities: The CPU time required ranged from 4.52 seconds to 139.5 seconds.

Held, Karp and Shareshian [39] approached the problem as a sequencing problem involving precedence relations that prohibit the occurrence of certain orderings. Their algorithm uses a dynamic programming approach to determine which feasible set of tasks to use for the assignment of tasks to stations. The procedure first enumerates all feasible subsets. A feasible subset is a subset of the tasks in a partially ordered set that may be executed in some order without the prior execution of any other task. The cost of a feasible subset is the minimum of the costs of its associated feasible subsequences. A feasible subsequence is a subsequence of the tasks that can be executed in the indicated order without any other tasks being done. Thus, each feasible subset may have several associated feasible subsequences. The cost of a feasible subsequence is composed of the cost of the feasible subsequence without the last task and the cost incurred by adding the last task to the set. That is:

$$\text{Cost of sequence } (A_1, \ldots, A_i, A_j) = \text{Cost of sequence } (A_1, \ldots, A_i) + \Delta(A_j)$$

where $\Delta(A_j) = t_j$, if A_j fits in the last station of $(A_1, \ldots, A_i)$

$\Delta(A_j) = t_j$ in the next station + idle time in the last station of $(A_1, \ldots, A_i)$, if A_j does not fit.

This relationship is used recursively to get the costs of the subsets with two tasks from those with one, then the costs of the subsets with three tasks from those with two, and so on, until the cost of the entire line is obtained. The advantage of this procedure is that only the feasible subsets and their costs must be saved. As the problem size increases, direct application of the algorithm requires excessive amounts of storage. Thus, an approximation is utilized. The set of tasks is partitioned into smaller groups of tasks and these groups replace the individual tasks in their recursive relationship. The authors managed to solve a 180-task problem with randomly generated task times and precedence relations in 5 to 7 minutes on the IBM 7090. It took 20 seconds for a 36-task problem and 24 minutes for a 612-task problem.

Kao [49] presented a dynamic programming approach for the problem with stochastic task performance times. The objective is to find a grouping of tasks that satisfies all precedence relations and minimizes the labor cost. The formulation assures that the probability that the resulting work content at each station is no more than the cycle time is bounded by a prespecified value. In other words, tasks are assigned to a minimum number of stations provided that at each station there is at least a given probability of completing the work within the cycle time.

He defines the work content of station n, S_n as follows: $S_n = \sum_{i \in ST_n} t_i$. Since t_i 's are random variables, S_n 's are also random variables. The goal is to assign tasks to a minimum number of stations while observing all precedence constraints and the constraints that $\Pr(S_n \le C) \ge \theta$ for all n, where θ is the given lower bound $(0 \le \theta \le 1)$. He defines feasible sets equivalent to feasible subsets in the formulation of Held, Karp and Shareshian [39]. The feasible sets are the states in the dynamic programming formulation.

The dynamic programming formulation can be described as follow: The return function associated with state S is $T(S) = (n, G_r)$ where n is the minimum number of stations needed to accommodate the tasks in S while observing all precedence relations and G_r is the distribution function for r, the sum of the task performance times assigned to the last station under an optimal grouping for all tasks in S. For any task $e \in S$ for which S - e is a state, let $T(S - e) = (m, G_s)$. Then, define;

$$\Delta(T(S - e), e) = (m, G_{s+e}) \text{ if } G_{s+e}^{-1}(\theta) \le C$$

$$\Delta(T(S - e), e) = (m + 1, F_e) \quad \text{otherwise}$$

where G_{s+e} is the distribution function for the random variable $S + t_e$. The Δ function corresponds to the placing of task e at the end of an optimal sequence for the state S - e in the following manner: Place task e in the last station for S - e if its inclusion does not violate the probability constraint on the station work content, otherwise create a new station to include it.

The author concludes that the procedure is useful only for problems of limited size due to the fact that storage and computation requirements grow very rapidly as the number of tasks in an assembly line increases. Kao [50] and Kao and Queyranne [51] later improved the computational aspects of the procedure in order to solve larger problems by utilizing efficient labelling, addressing and generation procedures.

Although extensive research has been done on the deterministic assembly line balancing problem, optimal solution procedures capable of solving realistically complex, large-scale problems within existing computer capabilities are nonexistent. Given the developments over the last three decades in improving the efficiency and applicability of the optimum seeking solution procedures, it seems unlikely that further developments along this front will succeed to gain widespread practical use. Barring any breakthroughs in computer computational efficiency, the heuristic procedures developed for the solution of the problem appear to offer the most useful research track. These heuristic procedures have the advantage of addressing realistically sized problems with the minimal computational requirements. Some of these heuristic procedures reported in the literature will be discussed in the next section.

2.1.2 Heuristics

Webster's Dictionary of the English Language defines the adjective "heuristic" as "involving or serving as an aid to learning, discovery or problem-solving by experimental and especially trial-and-error methods". A heuristic procedure utilizes principles or devices that contribute to reduction of search in problem-solving activity. Applying heuristic procedures to the assembly line balancing problem is very attractive from the computational point of view, on the other hand, the deviation from the global optimum solution usually loses the attractiveness of the procedures. Nevertheless, due to the immense computational complexity of the problem, the heuristic procedures appear to be more promising than the optimum seeking algorithms. There are several heuristic procedures developed and reported in the literature for the assembly line balancing problem, and they will be reviewed in this section. Most of the techniques assign weights to the tasks in the problem and those weights determine the task to be selected for assignment.

Hu [43] described a simple procedure to minimize cycle time given the number of stations subject to the very severe restriction that all task times are equal. For the problem defined, he developed lower bounds on the cycle time given the number of stations and the number of stations given the cycle time.

Helgeson and Birnie [40] have proposed a heuristic procedure called "ranked positional weight technique" (RPWT) that could result in near optimal solutions. The method's advantage is that the practitioner is provided very quickly with a decent balance for improvement within minimal computation time. In the method, each task is given a weight equal to the sum of its task time and the task times of all other tasks that follow it on the precedence diagram. Then, the tasks are listed in descending order of their weights and an attempt is made to assign tasks to stations in that order. If a task takes longer than the time remaining in the station or would violate the precedence constraints, the next task is tried until all the remaining tasks are searched. If no further task can be assigned to a station, the next station is opened. To provide two line designs to choose from, they proposed the idea of "inverse positional weight" which is obtained by looking at the assembly operation from the end to the start of the line. Assignment to stations starts from the last station and proceedes forward from there. Although the technique does not guarantee an optimal solution, it makes it possible to test many alternative balances by trying different cycle times with economical computer manipulation. In spite of the fact that the method is very popular in the literature and well accepted by the majority of the readers and practitioners, Ignall [44] reports that the method results in a solution far away from the optimum for his example problem. Mastor [65] also supports Ignall [44] showing that the technique performs worse than almost all of the other techniques compared in his study. Buxey [12] improved the technique with paralleling of stations.

Tonge [101,102] developed a heuristic procedure for the problem consisting of three phases: i) simplification of the initial problem by grouping adjacent tasks into compound tasks, ii) solution of the more simple problems by assigning tasks to stations at the least complex level possible, breaking up the compound tasks into their elements only when necessary for a solution, and iii) smoothing the resulting balance by transferring tasks among stations until the distribution of assigned time is as even as possible. The third phase had not been programmed when Tonge wrote his article; thus, comment on how well the procedure performed is impossible. For the first two phases, the procedure took approximately 11 minutes for the 11-task problem taken from Jackson [45], and 5 hours for a 70-task problem on a relatively slow JOHNNIAC computer.

Later, Tonge [103] proposed a procedure which assignes tasks to stations by randomly selecting a heuristic procedure for choosing the next task to be added to the current station. Based on three example problems and by trying different cycle times, he concludes that the random selection of heuristics for choosing the next task does as well as or better than, either using an individual heuristic procedure alone, or randomly choosing the tasks without intervening choice of heuristic procedures.

Kilbridge and Wester [53,54] proposed a technique developed primarily to balance lines without the aid of a computer. The main feature of their technique is to group tasks into columns in the precedence diagram where tasks are placed as far left as possible without violating the precedence constraints. In such a diagram, tasks can be permuted among themselves in each column and some of the tasks can be moved laterally from their columns to positions to their right without violating the precedence constraints. Then, two properties of the tasks in the diagram - permutability within columns and lateral transferability - are exploited in the attempt to achieve optimum balance. As Kilbridge and Wester [53] state, the technique is not a mere mechanical procedure, since a fair amount of judgement and intuition must be used to derive a meaningful solution. It is a simple, powerful technique especially for large cycle times, when one station crosses several columns. But, on the other hand, for low cycle times, where one column may require two or more stations, a fair amount of adjustment is necessary, with no guarantee of a good balance. Thus, the technique loses its attractiveness in such cases. The authors [54] applied the procedure to a problem taken from industry in which fixed facilities and positional restrictions exist. The authors [52] also examined the relation of balance delay with various problem parameters; range of task times, cycle time, degree of precedence relation flexibility. They suggest that balance delay is very sensitive to the right selection of the cycle time. Thomopoulos [99,100] extended Kilbridge and Wester's technique to apply to mixed-model problems and he presented a procedure to allocate tasks to stations by forming a combined precedence diagram of the models involved in the problem.

Hoffmann [42] developed an enumeration method which generates all feasible station assignments that do not exceed the cycle time and selects the best arrangement from among these by use of a triangular precedence matrix. The procedure selects as the first station that feasible subset of tasks that leaves the least idle time, then selects from the remaining tasks the subset that leaves the least idle time in the second station, and so on. Hoffmann [42] developed a FORTRAN program which can handle lines with up to 99 tasks. The program balanced 19 to 76-task lines in 3 to 10 minutes on the CDC 1604 computer. Although the method may be computationally very expensive, Gehrlein and Patterson [34] demonstrated that the method, suitably modified, could be used to solve problems of moderate sizes.

Moodie and Young [68] developed a two-phase heuristic procedure for lines with either constant or variable task time values. In the first phase, a preliminary balance is obtained by selecting the tasks with no unassigned predecessors and fit the remaining station time in the order of largest performance time. In the second phase, tasks are shifted between stations in an attempt to reduce idle time or a smoothness index and distribute the idle time equally to all stations. To deal with variable task time values, station times are computed as follows:

$$S_j = \sum_{i \in ST_j} \mu_i + r \sqrt{\sum_{i \in ST_j} \sigma_i^2}$$

where r is a multiplier. This approach produces a design which assures that at every station a predetermined probability of completion of the tasks is maintaned. As Freeman and Jucker [32] state, the motivation for this objective is not clear at all. They also considered trading off large and small variance tasks, so that both the average time summations and the variance summations are as equal as possible for all stations. They wrote a FORTRAN program which solved a 48-task problem with variable task times in 1.38 minutes, and a 70-task problem with constant task times in 1.20 minutes on an IBM 7090.

Sarker and Shanthikumar [87] developed a technique which is quite similar to the one developed by Moodie and Young [68]. The technique enables to balance lines where the task times might be greater than the cycle time.

Arcus [3] developed a technique called COMSOAL in which the main idea is the random generation of a feasible sequence. The technique assignes the same probability of selection to the tasks with no unassigned predecessors and fit the remaining station time. Judging on the basis of the yield of good balances, he explored other methods for weighting the tasks; in other words, a couple of methods of biasing the tasks available for selection were developed. Among the nine methods developed, the one which is a combination of the others gives the best results. Arcus [3] managed to solve a 1000-task problem with a known optimum of 200 stations with zero idle time, and the maximum possible number of tasks available for assignment being 69. Using about half of the capacity of an IBM 7094 computer, a sequence requiring 203 stations (1.48% idle time) was achieved in 2 minutes. Buxey [12] improved the technique further with paralleling of stations.

Nevins [69] developed a general purpose heuristic program and successfully applied it to the assembly line balancing problem. He called the procedure 'best bud search' which does not attempt to minimize the number of stations directly, rather an upper bound on the number of stations is imposed and the problem is solved for that many stations. If the attempt is successful, the number of stations is decremented by one, and another attempt is made until it is either impossible or computationally impractical to get a smaller number of stations. Nevins [69] tested the problems solved by Tonge [102] and obtained as good or better results.

Macaskill [63] presented a computer implemented assignment heuristic procedure to solve the mixed-model assembly line balancing problem. The procedure does not require excessive computer effort and storage, but as differences in model work content increase, the procedure results in reduced performance of the line, increased line length and increased sensitivity to the sequence in which product units are fed to the line.

Agrawal [1] developed a procedure which utilizes a decision rule for alloting the work to stations called "largest set rule". The procedure computes the cumulative time for each task which is the time for performing the task and all the tasks preceding it. Then, the largest cumulative time which is less than the cycle time is selected and the associated tasks are assigned to the worker. The procedure is repeated on the truncated precedence diagram until all the tasks are assigned. After the work is alloted to workers, the designer has to decide on the sequence in which these workers should be positioned on the line. He presented some objectives that can be pursued for sequencing the workers. Although the procedure is computer efficient, there is no apparent guarantee of the optimal solution. The procedure took 8.5 seconds for the 45-task problem reported by Kilbridge and Wester [54].

Pinto, Dannenbring and Khumawala [74] presented a heuristic network procedure which is based on Gutjahr and Nemhauser's [38] shortest route algorithm. They utilize other heuristic rules such as RPWT, largest task time, smallest task time, and random assignment to generate the nodes. The set of nodes generated are combined to form a composite network. The procedure took 9.4 seconds of CPU time on the IBM 360/75 for a 50-task problem with 10 stations and a balance efficiency of 96.8%.

Akagi, Osaki and Kikuchi [2] proposed a method which allows assigning more than one worker to a station. Tasks are assigned to stations according to a couple of rules reported in the literature. The procedure is repeated for different number of workers at each station. In the second phase of this two-phase technique, tasks are assigned to workers within each station.

Dar-El [20] developed a method which minimizes the cycle time for a given number of stations. The method starts with the minimum theoretical cycle time and proceeds with the generation of a feasible sequence of tasks which are grouped into station assignments. The method aims at grouping all the tasks into the required number of stations. If a feasible sequence can not be extended, the method applies a backtracking procedure which either partitiones the tasks correctly or results in an increase of one time unit of the cycle time. Dar-El [20] improved this method by imposing rules which limit the backtracking iterations. This method, called MALB, dominates COMSOAL developed by Arcus [3] and 10-SP (a method selecting the best of ten solutions, each obtained by using a different ranking system, e.g., as with RPWT) in the problems tested. Dar-El and Rubinovitch [25] developed another method which generates alternative solutions of equal quality. This method, called MUST, dominates or gives equal results with MALB in every case. Dar-El [21] compared MUST with COMSOAL and single-pass methods such as RPWT developed by Helgeson and Birnie [40]. MALB technique gives consistently superior results over the others. Dar-El and Cother [23] presented a heuristic procedure for the model sequencing problem for mixed-model lines in which the objective is to minimize the line length for zero worker idle time. The effects of demand deviation for each model, number of stations, number of models, model cycle time deviation and operator time deviation are analyzed, and the last three factors are found to be the major factors influencing the line length. Later, Dar-El and Cucuy [24] further improved the approach made by Dar-El and Cother.

Bennett and Byrd [8] presented a 'trainable heuristic procedure' which consists of two stages. In the first stage, the procedure is trained by accumulating experience on the effectiveness of several heuristic rules on small problems for which the optimum is known. In the second stage, the findings of the first stage are used to provide a near optimal solution which is fed to an optimization procedure as a starting point. The authors use several empirical values and rules with no apparent justification.

Rosenblatt and Carlson [82] developed a model for the problem in which the objective is to maximize $\frac{f_1}{C} - K f_2$, where f_1 and f_2 are contribution per unit of product and fixed cost per unit of time for using the Kth station, respectively. They show that maximizing the efficiency of a line might not necessarily maximize profit. The main idea of the solution procedure is to generate all feasible combinations of K and C. Some properties of the optimal solution are presented which could reduce the computational requirements.

Davis and Simmons [26] considered improving the line efficiency of an unbalanced line. By employing a dynamic programming - heuristic procedure, the stations which should be operated or kept idle at each cycle are determined, as well as the levels of in-process inventories between stations.

Chakravarty and Shtub [15,16] developed a technique for solving mixed-model, unpaced assembly line balancing problem. The performance measure includes labor cost, inventory holding cost, and setup cost. Different models are represented on a combined precedence diagram. They propose two procedures for grouping tasks to stations. First one is based on a shortest-path approach and the other one is similar to the RPWT developed by Helgeson and Birnie [40].

Driscoll and Abdel-Shafi [28] developed a simulation model to examine line balancing problem solutions and efficiency levels in circumstances where original balancing information has been subjected to variance. The parameters examined are cycle time, open versus closed stations and product mix in mixed-model lines.

2.2 Stochastic Assembly Line Balancing

With the relaxation of the deterministic task performance time assumption, several issues become relevant that complicate the analysis and the development of solution procedures. The solution procedures developed for the stochastic assembly line balancing problem are all heuristic procedures. Some of the procedures reported in the literature are discussed below.

Freeman and Jucker [32] mentioned the stochasticity of the task times and stated that the likelihood of a development of a solution technique is remote. Moodie and Young [68] considered the variability of task times by providing an allowance to the operator for a given confidence level of task completion. Arcus [3] and Wild [107] also mentioned the stochastic performance of the workers. Mansoor and Ben-Tuvia [64] introduced the concept of an incentive plan for assembly line workers. These attempts to solve stochastic assembly line balancing problems are all extensions of the techniques for deterministic lines. However, there are a few techniques specifically developed for solving stochastic lines reported in the literature, and they will be explained briefly in this section.

Buxey [13] examined stochastic assembly lines via Monte Carlo simulation and concluded that for good line designs, the ratio of on-line inventory to number of stations should be greater than unity. The simulation study also reveals that a criterion of maximizing output would imply the acceptance of a small proportion of unfinished units.

Kottas and Lau [57] developed a heuristic procedure for solving mainly the stochastic assembly line balancing problem. They assume that the time to complete any task i is normally distributed with mean μ_i and variance σ_i^2. Whenever a task is not finished, the unit goes down the line with as many of the remaining tasks being completed as possible. All unfinished tasks are completed off the line; the cost to complete the task off the line is not a function of what fraction of the task is completed on the line. Their procedure can be described as follows: An available list is formed by identifying the tasks with no unassigned predecessors. This list is updated each time a task is assigned. Then, a desirable list is formed by identifying the available list tasks which are marginally desirable for assignment. A task is considered marginally desirable when its anticipated labor savings in the specific position under consideration is larger than its expected incompletion cost. The tasks with virtual certainty of completion are assigned first in descending order of their incompletion costs. These tasks comprise the sure list. If the sure list is empty, then the desirable list tasks are assigned in the ascending order of their incompletion costs. When the desirable list gets empty, a new station is established. The tasks which are never marginally desirable are assigned as the first tasks in stations as soon as they are placed in the available list. The procedure continues until the available list gets empty. The procedure is computationally very attractive; up to 50-task lines have been balanced in under 0.1 seconds CPU time on an IBM 360/75. On the other hand, since the procedure is a single-pass technique, the solution found might be far from the optimal solution, so a lot of precaution should be taken when applying the procedure.

Since this technique will be refered to later, we illustrate it here using the example of Section 4.3. Table 1 summarizes the solution procedure of the example. Note that $Z_k = (C - \sum_{i \in U} \mu_i)/(\sum_{i \in U} \sigma_i^2)^{1/2}$ where U is the set of tasks assigned to a station. Z_k^* is the value below which the outcomes of a normally distributed random variable with mean 0 and standard deviation 1 have a $1 - \frac{L \mu_k}{CIC_k}$ probability of occurring. CIC_k is the cumulative incompletion cost of task k, and here it is defined to be equal to the summation of the incompletion cost of task k and those of the tasks following task k on the precedence diagram. Desirable list tasks are the available list tasks whose $Z_k \geq Z_k^*$, and sure list tasks are the desirable list tasks whose $Z_k > 2.575$. Critical list tasks are the available list tasks whose $Z_k < Z_k^*$ when the station is empty. The technique generates the solution shown below which is the optimal solution of the problem as will be indicated in Section 4.3.

Tasks of station 1 : 1,2
Tasks of station 2 : 3
Tasks of station 3 : 4

Kottas and Lau [58] later developed a procedure for evaluating the expected incompletion cost of a paced line. The procedure identifies all the possible combinations of incomplete tasks. A task is considered incomplete if either the time needed exceeds the time available at a station, or a previous task has not been completed. The summation of the products of incompletion probabilities and incompletion costs of the combinations constitute the expected incompletion cost of the line.

Kottas and Lau [59] developed another heuristic procedure which generates several promising line designs. The approach is conceptually related to the techniques of Arcus [3] and Tonge [103]. First, several designs are generated with a modified version of their earlier procedure [57]. Several selection rules are used for the desirable list tasks. Then, the dominated designs are eliminated and the remaining ones are evaluated with their evaluation technique [58]. The design with the lowest cost

Table 1. Kottas and Lau technique solution of the example problem of Section 4.3

Station	Assigned Tasks	Available List Tasks	Z_k	Z_k^*	Desirable List Tasks	Sure List Tasks	Critical List Tasks
1	None	1	6.71	2.05	1	1	-
	1	2 3	3.65 < 0	1.96 1.58	2 -	2 -	- -
	1,2	3	< 0	1.58	-	-	-
2	None	3	1.58	1.58	3	-	-
	3	4	< 0	1.28	-	-	-
3	None	4	4.00	1.28	4	4	-
	4						

is the solution to the procedure. Twenty-four randomly generated problems, with sizes ranging from 50 to 80 tasks, were solved; each took 12 to 240 seconds of CPU time on an IBM 370/155 during which 300 to 500 line designs were generated and of these the expected incompletion cost of 40 to 100 designs were evaluated.

Vrat and Virani [105] applied Kottas and Lau's [57] technique to a real life problem. They modified the technique to enable it to tackle task times greater than the cycle time by paralleling of stations for such tasks. They redesigned a line with 19 tasks; the expected unit cost of the new design is anticipated to be 26.0% less than the current value. They also compared the solutions obtained by trying different cycle times with the technique of Moodie and Young [68] for 95% or more probability of completion at each station and obtained a lower total operating cost for each case.

Shtub [93] presented an heuristic procedure for designing lines with stochastic task times and multiple manning of stations. The procedure is quite similar to Kottas and Lau's [57] technique. Formation of the available and fit (desirable) lists is identical. Tasks are selected from the fit list randomly. An estimate of the number of subsequent stations is made after establishing each station, and the number of operators at the current station is determined accordingly. The approach is very attractive from a computational point of view, but it has the drawbacks of Kottas and Lau's [57] approach. Twenty-seven 30-task problems with 3 possible number of workers at each station were solved on an AMDAHL 470/V8 system, and each problem took on the average 2.72 seconds of CPU time.

Reeve and Thomas [79] compared four solution procedures for stochastic assembly lines. The first procedure is the Trade and Transfer concept introduced by Moodie and Young [68]. One-for-one task trades between stations are attempted in order to reduce the probability of exceeding the cycle time. The second procedure is the Branch and Bound technique described by Reeve [79]. The third procedure is an extension of the previous one with some heuristic rules. And the last procedure, BABTAB, is a combination of the first and the third procedures. They tested four problems and indicated that the Branch and Bound technique guarantees a global optimum with excessive computer time. If an adequate supply of computer time is available, the Heuristic Branch and Bound technique gives very good results. BABTAB yields good results in relatively short time periods. On the other hand, the conclusions reached are not justified, since the number of example problems solved is very small.

Sculli [91] considered adjusting the line design after the initial design because of the several dynamic factors involved in the process. The objective of assigning workers to the last station is to meet demand, and for intermediate stations the aim is to keep the following stations working. The output rate at station i is assumed to be distributed $N(r_i, p_i^2 r_i^2)$. Workers are assigned to stations in such a way that the probability of starvation of any station is less than a given value. The technique also computes the average in-process inventories. Later, Sculli [92] developed a computer program which finds a compromise solution to the line balancing problem by interacting with the user. The constraints involved in the procedure are that the output from a station should be sufficient to keep the following station working, and the number of workers assigned to a station must not exceed a given upper bound. The program attempts to assign workers in an optimal manner; if a solution is not possible, options to change output rate or other constraints are offered, so that a compromise solution could be found. The program does not have any academic interest, but has a practical value; as Sculli [92] says: "It is intended for use by the line manager to determine the allocation of operators at the start of each shift or work period".

3.0 Development Of The Cost Model

In this chapter, we develop a general expression which captures the cost factors of the objective function stated in Chapter 1. The variables that should be determined in order to compute the terms of the objective function are discussed. The derivation of each variable is also presented. The assumptions of the model and the problem parameters are discussed in Section 3.1. The development of the model is presented in Section 3.2.

3.1 Assumptions Of The Model and Problem Parameters

The formulation of the problem is developed based on the following assumptions.

1. Task performance times are normally distributed random variables with known means and variances. They are truncated at zero and are independent of each other and the ordering of tasks in a station.

2. The precedence diagram of the problem is known with certainty and it is the most efficient diagram possible for the problem.

3. No splitting of the tasks among stations is permitted.

4. Each station is manned by one worker who is paid the same wage regardless of his assignment.

5. Demand rate is known with certainty.

6. No buffer inventory between the stations is allowed.

7. The precedence constraints are the only restrictions on making station assignments.

8. Each task can only be started if all its predecessors have been completed.

9. Whenever a task is not finished, the unit moves down the line with as many of the remaining tasks being completed as possible.

10. Uncompleted tasks are completed off the line; the cost to complete a task off the line is not dependent on the fraction of the task completed on the line.

11. Incompletion of a task does not affect the rate at which units are moved through the line.

Assumption 1 is similar to that made considering task time variations by others in the literature [16,49,50,57,58,59,68,93,105]. The conditions under which this assumption is justified are developed below. Assumption 2 is made since task definitions and the precedence diagram of the problem should be determined before the solution procedure is applied to the line balancing problem. Assumptions 2, 3, 4, 5, 6, 7 and 8 are similar to those made in the majority of the line balancing literature. However, the formulation could be easily extended to relax assumptions 4 and 7; more than one worker could be assigned to a station, nonidentical wage rates could be applied for different tasks, and other constraints, such as zoning or positional constraints, could be imposed on making station assignments. Assumptions 9 and 10 represent only one of the ways the incompletion situations are handled; the formulation could be modified to handle other incompletion situations, such as the incompleted units are scrapped.

The truncation of the task performance time distributions at zero can be made if the probability that a normally distributed random variable can take negative values is small enough. Next, we develop some conditions under which this is true. To that end, let E represent the area to the left of zero under a Normal distribution with mean μ_i and variance σ_i^2. This area is depicted in Figure 3. Let ε be a small quantity greater than zero. If $\Phi(.)$ is the cumulative standard Normal distribution function and $\sigma_i = a \times \mu_i$ for all i, then the desired condition is as follows:

$$E = \Phi\left(\frac{-\mu_i}{\sigma_i}\right) \leq \varepsilon \qquad \text{for } i = 1,\ldots\ldots,N$$

The above condition reduces to the following expression:

$$a \leq -\frac{1}{\Phi^{-1}(\varepsilon)}$$

In other words, the truncation of the Normal distribution can be ignored if a is smaller than the above value determined as a function of ε. Table 2 depicts the upper bounds on the value of a for different ε values. The ε value column of Table 2 gives the area under the Normal distribution function to the left of zero that is discarded. For example, for $\varepsilon = 0.05$, we assume that when the area under the Normal distribution function to the left of zero is equal to or less than 0.05, it is negligible. The second column of the Table depicts the upper bounds on the values of a. In other words, a should be smaller than the value given in the column in order to make the area under the Normal distribution function to the left of zero less than or equal to the corresponding ε value. Note that for practical task performance times, $a \leq 1.0$, since $a > 1$ implies that the standard deviation of a task performance time is greater than its expected value, and this situation is highly improbable for tasks of realistic assembly lines. Typically, a is much small than 1.0, specifically,

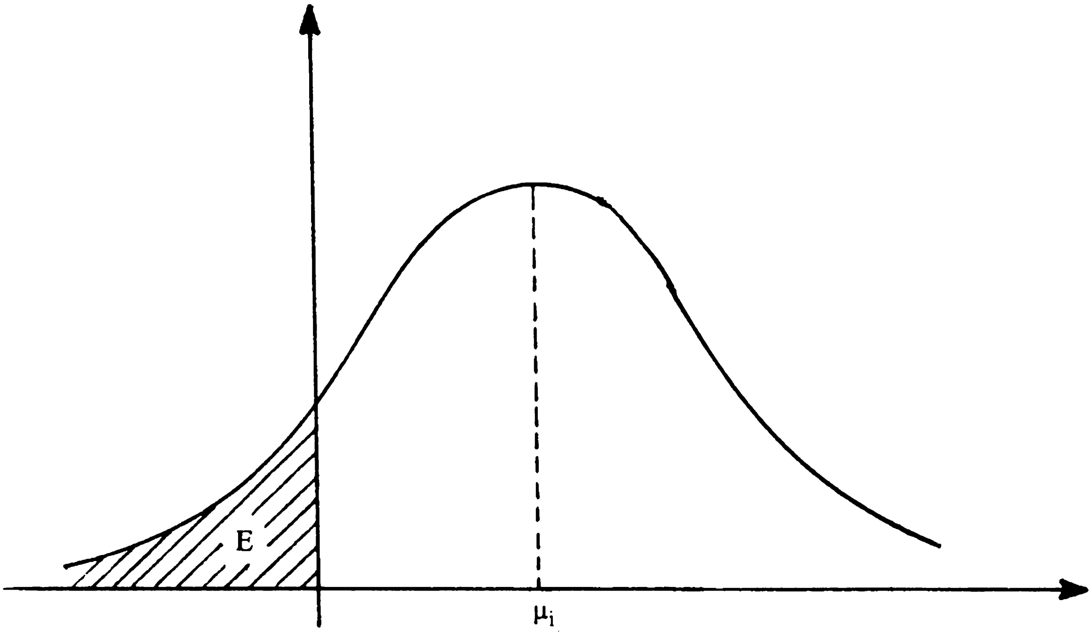


Figure 3. Probability of a normally distributed random variable taking negative values

around 0.2 [109], and consequently it can be safely assumed that the effect of truncating the Normal distribution at zero is negligible.

Table 2. Upper bounds on the values of a for different ε values

ε	a
0.20	1.188
0.10	0.780
0.05	0.608
0.03	0.364
0.01	0.324

Incompletion costs are calculated as if the incomplete tasks are handled by a group of workers supporting the line. Incompletion cost of task i, IC_i is assumed to be larger than $L \times \mu_i$ for $i = 1,\dots,N$, where L is the labor rate.

The demand rate for the product is assumed to be known with certainty. A fixed demand rate imposes a fixed cycle time. The model will be developed to give a solution for the cycle time imposed by the demand rate.

3.2 Development of the Model

The objective function of the single-model, stochastic assembly line balancing problem stated in Chapter 1 is as follows:

Min Z = Total Labor Cost + Total Expected Incompletion Cost

In this section, we develop a general expression that captures the cost terms of the above objective function for a given number of stations and allocations of tasks to these stations. First, we introduce some notation and clarify the use of the notation with an example. Then, the variables which should be determined for each task in order to compute the total expected incompletion cost term are discussed. These variables include the probability that the task can be started to be processed, the probability that the task is not completed within C after it has been started to be processed, and the cost incurred due to the incompletion of a task. The derivation of these variables are also discussed and clarified with an example.

Let the set of tasks following task i on the precedence diagram and in the station which contains task i be denoted by A'_i and H_i , respectively. Note that when task i is not completed within C, then the tasks in H_i cannot be started. Moreover, the tasks in $\bigcup_{j \in H_i} A'_j$ also cannot be started. Let $A_i = A'_i \cup H_i \cup (\bigcup_{j \in H_i} A'_j)$. Hence, incompletion of task i incurs a cost equal to the incompletion cost of task i and that of the tasks in A_i ; that is $\sum_{j \in A_i} IC_j + IC_i$. Let P_i and B_i be the set of tasks preceding task i on the precedence diagram and in the station which contains task i, respectively. Task i can be started only if the tasks in P_i and the tasks in B_i that can be started are completed. The determination of P_i and A_i will be clarified with an example. Let the allocation of the tasks to stations of the precedence diagram depicted in Figure 1 be as follows:

Sequence of the tasks in station 1 : 1, 3
Sequence of the tasks in station 2 : 2, 4
Sequence of the tasks in station 3 : 5, 6, 7, 8
Sequence of the tasks in station 4 : 9, 10
Sequence of the tasks in station 5 : 11

The allocation of the tasks to stations is also depicted in Figure 4. Consider task 2 which is assigned to the second station as the first task. Referring to Figure 4, $A'_2 = \{6, 8, 10, 11\}$. Since $H_2 = \{4\}$, if task 2 is not completed within C, the tasks in $\bigcup_{j \in H_2} A'_j = \{7, 9, 11\}$ cannot be started. Therefore, $A_2 = \{4, 6, 7, 8, 9, 10, 11\}$. On the other hand, note that task 2 can be started only if task 1 is completed; that is $P_2 = \{1\}$. The determination of P_i and A_i for $i = 1,\ldots,11$ of the example are depicted in Table 3. Note that W denotes the set of all tasks in the problem.

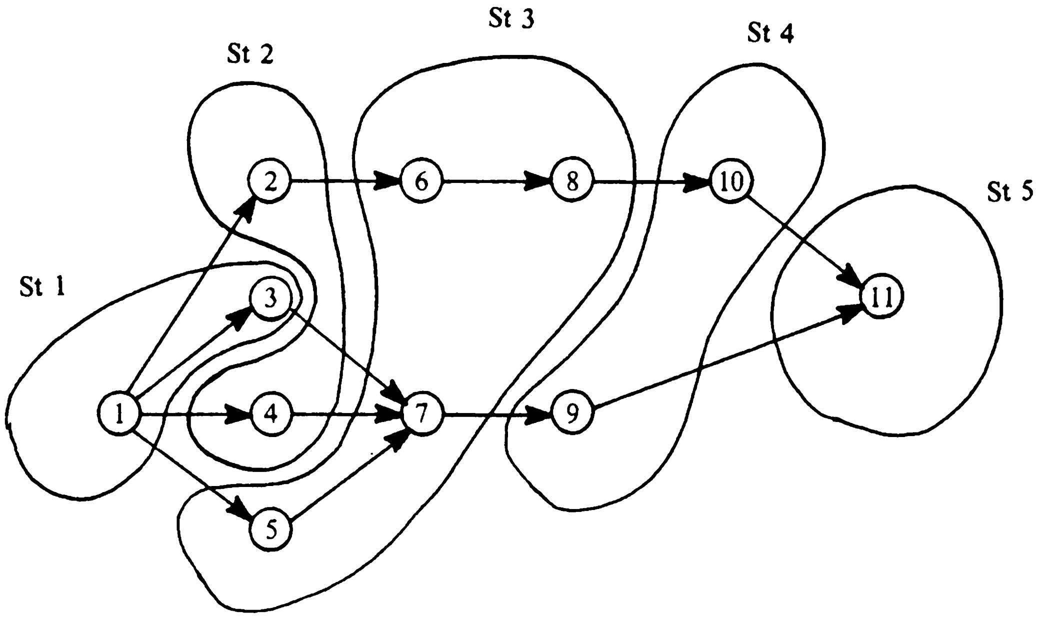


Figure 4. Allocation of tasks to stations of the example

Table 3. Determination of various variables to compute the total system cost of the example

Task, i	H_i	A'_i	$\bigcup_{j\in H_i} A'_j$	A_i	B_i	P_i
1	3	W - {1}	7,9,11	W - {1}	$\emptyset$	$\emptyset$
2	4	6,8,10,11	7,9,11	4,6,7,8,9,10,11	$\emptyset$	1
3	$\emptyset$	7,9,11	$\emptyset$	7,9,11	1	1
4	$\emptyset$	7,9,11	$\emptyset$	7,9,11	2	1
5	6,7,8	7,9,11	8,9,10,11	6,7,8,9,10,11	$\emptyset$	1
6	7,8	8,10,11	9,10,11	7,8,9,10,11	5	1,2
7	8	9,11	10,11	8,9,10,11	5,6	1,3,4,5
8	$\emptyset$	10,11	$\emptyset$	10,11	5,6,7	1,2,6
9	10	11	11	10,11	$\emptyset$	1,3,4,5,7
10	$\emptyset$	11	$\emptyset$	11	9	1,2,6,8
11	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	W - {11}

Next, we discuss the variables which should be determined for each task in order to compute the cost terms of the objective function. To that end, we derive the probability that a task is not completed within C after it has been started to be processed. Before developing an expression for the incompletion probability of a task, we first derive the probability that a task can get started to be processed.

Let b_i denote the station to which task i is assigned, and assume that task i is the second task in station b_i and $B_i = \{j\}$. In addition, assume $j \notin P_i$. Then, task i can be started if (i) task j is started and completed within C, or (ii) task j cannot be started, because a task in P_j is not completed in a previous station. These events will be called the starting events for task i. To extend the example a little further, assume that task i is the third task in station b_i and $B_i = \{j, k\}$. In addition, assume $j, k \notin P_i$. Then the starting events for task i are (i) tasks j and k can both be started and completed within C, (ii) task j cannot be started and task k can be started and completed within C, (iii) task k cannot be started and task j can be started and completed within C, (iv) both tasks j and k cannot be started. Note that if there are n tasks in B_i that do not belong to P_i , then there can be at most 2^n starting events for task i. Let T_i be the set of tasks in B_i that do not belong to P_i . Then, $T_i = B_i \cap (W - P_i)$. Let T_i^j be the jth starting event of task i in which the tasks in TS_i^j and task i can be started and the tasks in TN_i^j cannot be started. Note that $\{ B_i \cap P_i \} \subseteq TS_i^j$, and $B_i = TS_i^j \cup TN_i^j$ for $j = 1, \ldots\ldots, 2^n$. Let γ_i^j denote the probability that T_i^j occurs, and β_i^j denote the probability that T_i^j occurs and task i is incomplete while the tasks in TS_i^j are completed within C. Then, β_i^j can be expressed as follows:

$$\beta_i^j = \gamma_i^j \times \left[\Pr \{ \text{task i incomplete and tasks in } TS_i^j \text{ complete} \} \right]$$

To compute Pr { task i incomplete and tasks in TS_i^j complete }, let X and Y be the events that task i is not completed within C and tasks in TS_i^j are completed within C, respectively. Then, $\Pr \{X \text{ and } Y\} = \Pr \{X/Y\} . \Pr \{Y\}$.

Now, $\Pr \{X\} = \Pr \{X/Y\} . \Pr \{Y\} + \Pr \{X/\overline{Y}\} . \Pr \{\overline{Y}\}$, where $\overline{Y}$ is the complement of Y.

$$\text{Therefore, } \Pr \{X/Y\} = \frac{\Pr \{X\} - \Pr \{X/\overline{Y}\} \Pr \{\overline{Y}\}}{\Pr \{Y\}}$$

Note that $\Pr \{X/\overline{Y}\} = 1$, because this represents the probability that task i is incomplete given that the tasks in TS_i^j are incomplete (and the tasks in TS_i^j precede task i in the station). Hence, $\Pr \{X \text{ and } Y\} = \Pr \{X\} - \Pr \{\overline{Y}\}$. Then, β_i^j can be expressed as follows:

$$\beta_i^j = \gamma_i^j \times \left[\Pr \{ \text{task i incomplete} \} - \Pr \{ \text{tasks in } TS_i^j \text{ incomplete} \} \right]$$

$$= \gamma_i^j \times \left[\Pr \left\{ \sum_{k \in TS_i^j} t_k + t_i > C \right\} - \Pr \left\{ \sum_{k \in TS_i^j} t_k > C \right\} \right]$$

$$= \gamma_i^j \times \Gamma_i^j$$

$$\text{where, } \Gamma_i^j = \Pr \left\{ \sum_{k \in TS_i^j} t_k + t_i > C \right\} - \Pr \left\{ \sum_{k \in TS_i^j} t_k > C \right\}$$

Task performance times are assumed to be Normally distributed random variables with known means and variances. Since they are independent of each other and of the ordering in stations, then Γ_i^j can be expressed as follows:

$$\Gamma_i^j = \left[1 - \Phi\left(\frac{C - \left[\sum\limits_{k \in TS_i^j} \mu_k + \mu_i \right]}{\sqrt{\sum\limits_{k \in TS_i^j} \sigma_k^2 + \sigma_i^2}} \right) \right] - \left[1 - \Phi\left(\frac{C - \left[\sum\limits_{k \in TS_i^j} \mu_k \right]}{\sqrt{\sum\limits_{k \in TS_i^j} \sigma_k^2}} \right) \right]$$

Let β_i denote the probability that task i is started to be processed and not completed within C. If there are n tasks in T_i, then, as discussed before, there can be at most 2^n different starting events of task i, and β_i can be expressed as follows:

$$\beta_i = \sum_{j=1}^{2^n} \beta_i^j = \sum_{j=1}^{2^n} \gamma_i^j \times \Gamma_i^j$$

Consider tasks x and y such that $x \in TS_i^j$ and $y \in TN_i^j$. For T_i^j to occur, at least one task in P_y should not be completed within C. If $P_y \subseteq P_x$, then T_i^j is an infeasible starting event, since all the tasks in P_x should be completed within C. Let fs_i denote the number of feasible starting events for task i. Note that $fs_i \leq 2^n$, where n is the number of tasks in T_i. Then, β_i can be expressed as follows:

$$\beta_i = \sum_{j=1}^{fs_i} \beta_i^j$$

In the computation of β_i, the determination of Γ_i^j is straightforward once the assignment of tasks to a station is known. However, the determination of γ_i^j is not as straightforward because of the complexity of its occurrence. Here, we discuss a procedure to compute γ_i^j that uses a special enumeration tree. All possible ways of realizing the starting event T_i^j are represented with this tree. The cases in which the starting event T_i^j can be realized are represented, and the occurrence probabilities of these cases are computed. Consequently, the occurrence probability of the starting event T_i^j, γ_i^j is derived. Let $w_{i,k}^j$ denote the probability of occurrence of the kth case in the enumeration tree constructed to compute γ_i^j, and let $IW_{i,k}^j$ and $CW_{i,k}^j$ denote the sets of tasks that are incomplete and complete in the kth case of the tree, respectively. Let q_{k_i} denote the number of tasks assigned to station k_i. The tree has $b_i - 1$ levels and each level has q_{k_i} sublevels for

$k_l = 1, \ldots\ldots, b_l - 1$. The levels and sublevels represent the stations and the tasks assigned to the stations, respectively. The construction of the tree will be discussed with the example given above. The tree constructed to compute γ_8^k , where $TS_8^k = \{5, 6\}$ and $TN_8^k = \{7\}$, is depicted in Figure 5. For starting event T_8^k to occur, one or more tasks in P_7 should not be completed, and all the tasks in P_5, P_6 and P_8 should be completed. Note that $T_8 = \{5, 7\}$ and $B_8 \cap P_8 = \{6\}$. Note also that $P_5 = \{1\}$, $P_6 = \{1, 2\}$, $P_7 = \{1, 3, 4, 5\}$ and $P_8 = \{1, 2, 6\}$.

The numbers outside the nodes are the node numbers and the ones inside the nodes represent the tasks in the nodes. Nodes with tasks i and $\bar{i}$ represent that task i is completed or not completed within C, respectively. Level 1 corresponds to station 1 and the first sublevel of level 1 represents task 1. Task 1 is either completed or not completed within C, and these events are represented by nodes 1 and 2. If task 1 is completed, then task 3 can be started, and task 3 is either completed or not completed within C, and is represented by nodes 3 and 4. Note that for starting event T_8^k to occur, task 1 has to be completed, since $1 \in P_5, P_6, P_8$; thus, node 2 is pruned. Level 2 represents the second station. Task 2 can be started whether task 3 is completed or not, since $3 \notin P_2$. Task 2 is either completed or not completed within C, as represented by nodes 5, 6, 7 and 8. If task 2 is completed, then task 4 can be started. On the other hand, task 2 has to be completed for starting event T_8^k to occur, since $2 \in P_6, P_8$. Thus, nodes 6 and 8 are pruned and not branched into descendent nodes. Nodes 5 and 7 are branched into nodes representing task 4 being completed or not completed within C. Note that for starting event T_8^k to occur, one or more tasks in P_7 has to be incomplete; thus, node 9 represents an infeasible case, since none of the tasks in P_7 are incomplete in the case represented by node 9. The cases represented by nodes 10, 11 and 12 are the all possible cases for T_8^k to occur. The tasks that are complete and incomplete in these three cases are given below:

$CW_{8,1}^k = \{1, 2, 3\}$ and $IW_{8,1}^k = \{4\}$ corresponding to node 10

$CW_{8,2}^k = \{1, 2, 4\}$ and $IW_{8,2}^k = \{3\}$ corresponding to node 11

$CW_{8,3}^k = \{1, 2\}$ and $IW_{8,3}^k = \{3, 4\}$ corresponding to node 12

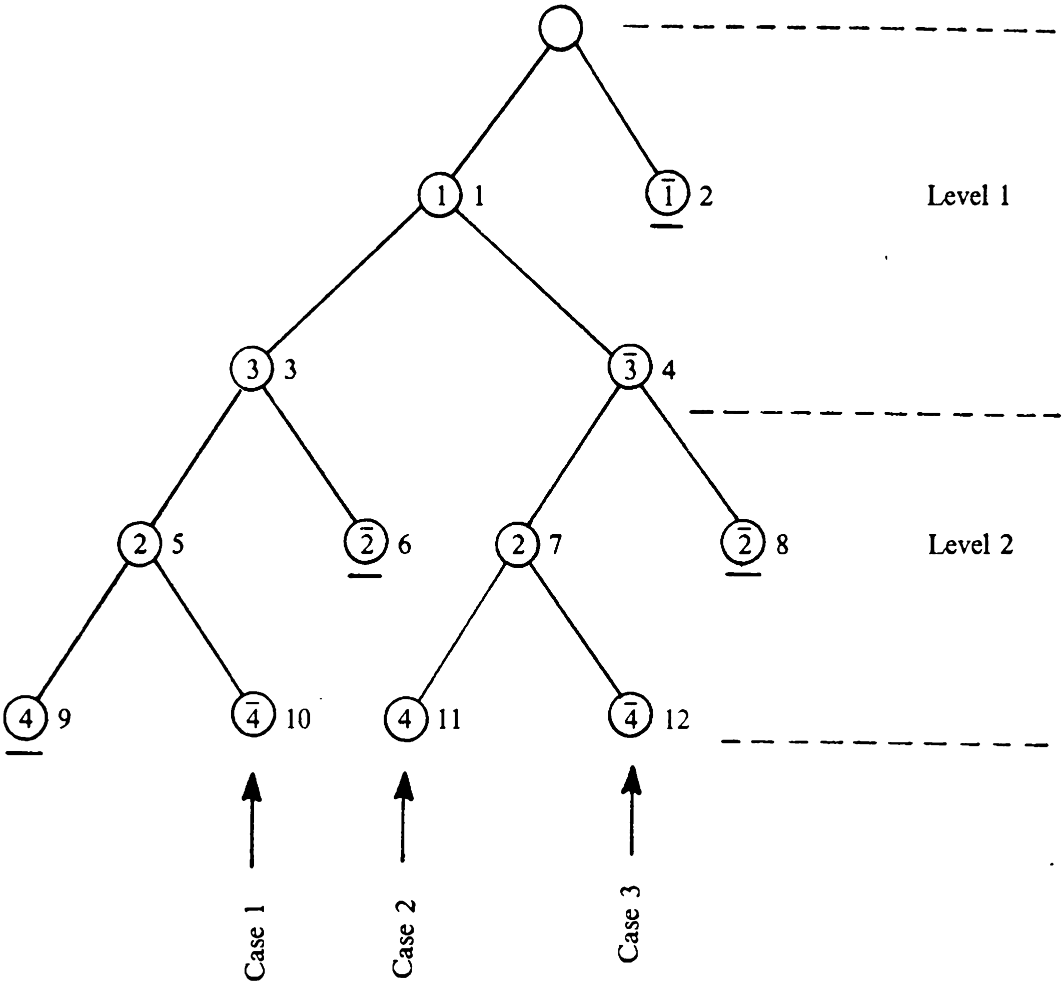


Figure 5. Probability enumeration tree to compute γ_8^k of the example

In the enumeration tree discussed above, a node at sublevel k with task j is branched into two descendent nodes at sublevel $k + 1$ with task i assigned to them if all the tasks in P_i are completed in the case represented by the parent node at sublevel k. Otherwise, if the parent node represents a case in which one or more tasks in P_i are incomplete, then the node is not branched into nodes at sublevel $k + 1$; sublevel $k + 1$ is skipped. Within a level, if the parent node represents a task being not completed, and if the node can be branched into nodes of the next sublevel, then the parent node is branched into a descendent node representing the task being incomplete. A node is pruned if the task represented by the node is incomplete, and that task is required to be completed for the associated starting event to occur. These cases will be further clarified with the following example. Consider task 10 in station 4 of the example above. Task 10 has two starting events; namely, task 9 can be started and task 9 cannot be started. Let's consider the second starting event; that is $TS^2_{10} = \emptyset$ and $TN^2_{10} = \{9\}$. Note that for starting event T^2_{10} to occur, one or more tasks in P_9 should not be completed, and all the tasks in P_{10} should be completed. Note also that $P_9 = \{1, 3, 4, 5, 7\}$ and $P_{10} = \{1, 2, 6, 8\}$.

The tree constructed to compute γ^2_{10} is depicted in Figure 6. The tree has 3 levels representing the first 3 stations on the line $(b_{10} - 1 = 3)$, and the levels have 2, 2 and 4 sublevels, respectively, representing the tasks assigned to the first 3 stations. Node 2 is pruned, since $1 \in P_{10}$. Note that all the tasks in P_{10} should be completed and one or more tasks in P_9 should not be completed in order to realize starting event T^2_{10} . Nodes 6 and 8 are pruned, since $2 \in P_{10}$. In the same token, nodes 22, 23, 25, 26, 28, 29, 31 and 32 are pruned, since $6 \in P_{10}$. Node 21 is branched into nodes 33 and 34 at sublevel 3, since the case represented by node 21 does not have any tasks in P_7 (task 7 is assigned to nodes 33 and 34) that are incomplete. On the other hand, node 24 cannot be branched into nodes of sublevel 3, because node 24 represents a case in which task 4 is incomplete and $4 \in P_7$. However, node 24 can be branched into nodes 38 and 39 at sublevel 4, because $4 \notin P_8$ (task 8 is assigned to nodes 38 and 39). The same situation applies to nodes 27 and 30. Node 34 is not branched into two descendent nodes, but into one descendent node (node 37), since node 34 represents task 7 being incomplete, and task 8 will also be incomplete because task 8 fol-

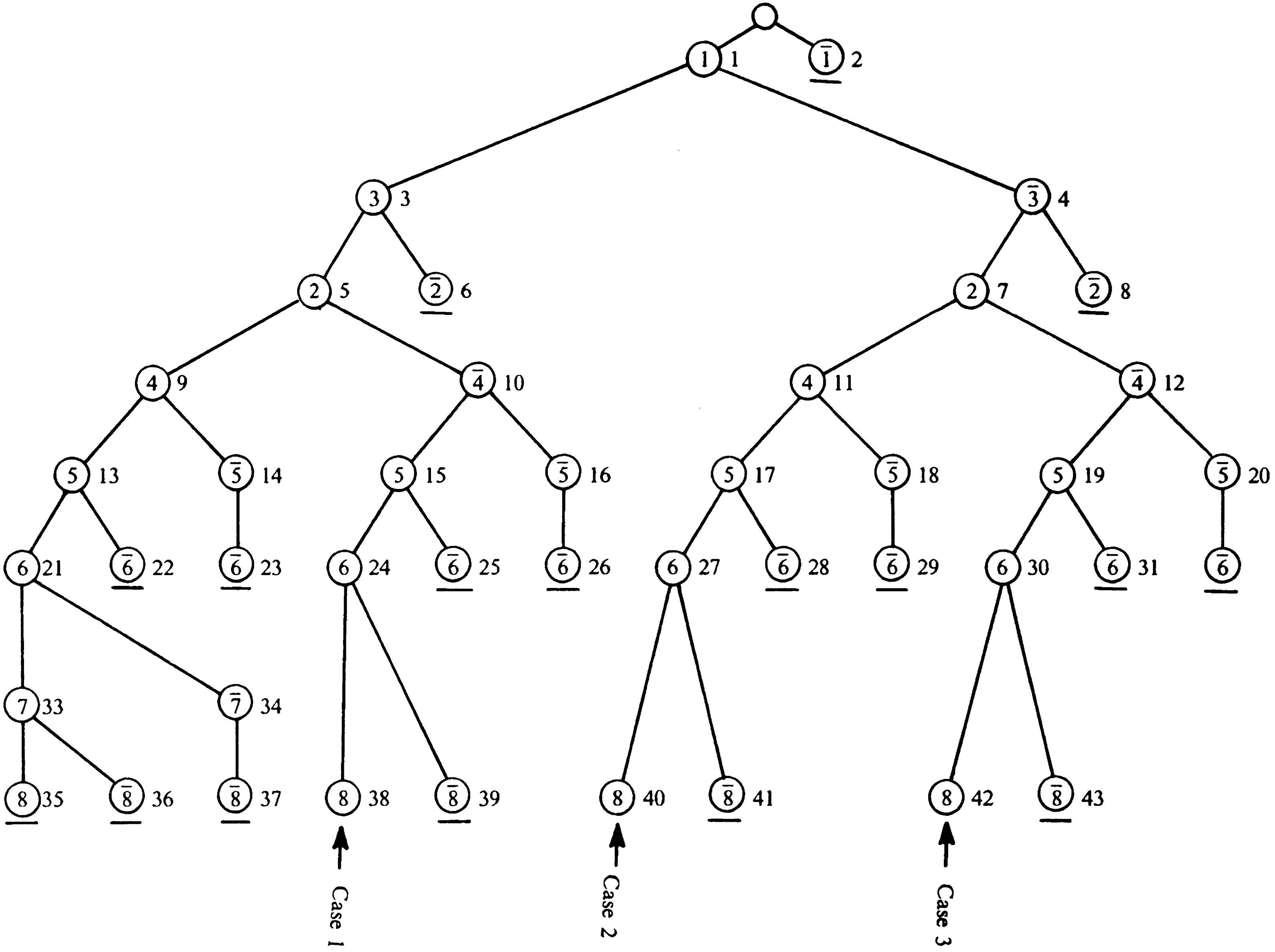


Figure 6. Probability enumeration tree to compute γ_{10}^2 of the example

lows task 7 in the station. Nodes 36, 37, 39, 41 and 43 are pruned, since $8 \in P_{10}$. Node 35 is also pruned, since the case represented by node 35 has all the tasks completed, whereas in order to realize starting event T_{10}^2, one or more tasks in P_9 should be incomplete; thus, node 35 represents an infeasible case. The cases represented by nodes 38, 40 and 42 are all the possible cases for starting event T_{10}^2 to occur. The tasks that are complete and incomplete in these three cases are given below:

$CW_{10,1}^2 = \{1, 3, 2, 5, 6, 8\}$ and $IW_{10,1}^2 = \{4, 7\}$ corresponding to node 38
$CW_{10,2}^2 = \{1, 2, 4, 5, 6, 8\}$ and $IW_{10,2}^2 = \{3, 7\}$ corresponding to node 40
$CW_{10,3}^2 = \{1, 2, 5, 6, 8\}$ and $IW_{10,3}^2 = \{3, 4, 7\}$ corresponding to node 42

Next, the computation of the occurrence probabilitites of the nodes in the enumeration tree will be discussed. Let o_k and $o_{\bar{k}}$ represent the occurrence probabilities of a pair of descendent nodes with task k assigned. Note that this pair of nodes are branched from a common parent node and let o_l denote the occurrence probability of the parent node. Then, $o_l = o_k + o_{\bar{k}}$. The occurrence probabilities of the nodes are computed in a similar manner as the computation of Γ_1^1. The occurrence probabilities of the nodes in the enumeration tree depicted in Figure 5 are computed as follows:

$$o_1 = \Phi\left(\frac{C - \mu_1}{\sigma_1}\right) \text{ and } o_2 = 1 - o_1$$

$$o_3 = \Phi\left(\frac{C - (\mu_1 + \mu_3)}{\sqrt{\sigma_1^2 + \sigma_3^2}}\right) \text{ and } o_4 = o_1 - o_3$$

$$o_5 = o_3 \times \Phi\left(\frac{C - \mu_2}{\sigma_2}\right) \text{ and } o_6 = o_3 - o_5$$

$$o_7 = o_4 \times \Phi\left(\frac{C - \mu_2}{\sigma_2}\right) \text{ and } o_8 = o_4 - o_7$$

$$o_9 = o_3 \times \Phi\left(\frac{C - (\mu_2 + \mu_4)}{\sqrt{\sigma_2^2 + \sigma_4^2}}\right) \text{ and } o_{10} = o_5 - o_9$$

$$o_{11} = o_4 \times \Phi\left(\frac{C - (\mu_2 + \mu_4)}{\sqrt{\sigma_2^2 + \sigma_4^2}}\right) \text{ and } o_{12} = o_7 - o_{11}$$

Now we can express the occurrence probability of starting event T_8^k, γ_8^k as the summation of the occurrence probabilities of nodes 10, 11 and 12. That is, $\gamma_8^k = o_{10} + o_{11} + o_{12} = w_{8,1}^k + w_{8,2}^k + w_{8,3}^k$.

With the enumeration tree discussed above, all possible cases for the starting event T_i^j to occur are enumerated, and the occurrence probability of each case is computed. If there are fw_i^j cases in the tree, then the occurrence probability of the starting event T_i^j, γ_i^j can be expressed as follows:

$$\gamma_i^j = \sum_{k=1}^{fw_i^j} w_{i,k}^j$$

Next, the contribution of task i to the total expected incompletion cost is derived. Before deriving the expression for the contribution of task i to the total expected incompletion cost, consider again task 10 in station 4 of the example above. The enumeration tree associated with the second starting event of task 10 was depicted in Figure 6. Consider the first case represented by node 38 where only tasks 4 and 7 are incomplete and all the other tasks are complete. This case occurs with probability $o_{38} = w_{10,1}^2$ and incurs an expected cost of $w_{10,1}^2 (IC_4 + IC_7 + IC_9 + IC_{11})$. The probability that this case of the starting event T_{10}^2 occurs and task 10 is not completed within C is $w_{10,1}^2 \times \Gamma_{10}^2$. The incompletion cost of the tasks in $A_{10} = \{11\}$ are multiplied with this probability and added to the total expected incompletion cost term. Note that the incompletion cost of task 11 is overcounted with probability $(w_{10,1}^2 \times \Gamma_{10}^2)$. Thus, for the case represented by node 38, $IC_{11}(w_{10,1}^2 \times \Gamma_{10}^2)$ is overcounted and it should be subtracted from the total expected incompletion cost term to obtain the exact value. The set of tasks whose incompletion costs are overcounted corresponding to the kth case of the starting event T_i^j is $CF_{i,k}^j = A_i \cap (\bigcup_{m \in IW_{i,k}^j} A_m)$. Note that the incompletion costs of the tasks in $CF_{i,k}^j$ are overcounted with probability $(w_{i,k}^j \times \Gamma_i^j)$. Let

SB_i^j denote the overcounted incompletion costs corresponding to T_i^j , then SB_i^j can be expressed as follows:

$$SB_i^j = \sum_{k=1}^{fw_i^j} \Gamma_i^j \times w_{i,k}^j \times \left[\sum_{m \in CF_{i,k}^j} IC_m \right]$$

Now, the expression for the contribution of task i to the total expected incompletion cost term can be defined. When task i is started and not completed within C, then this event causes task i and the tasks in A_i to be incomplete. In other words, the cost incurred due to the incompletion of task i is $(IC_i + \sum_{j \in A_i} IC_j)$, and the expected incompletion cost of task i is $\beta_i \times (IC_i + \sum_{j \in A_i} IC_j)$.

The computation of the total labor cost term of the objective function for a given number of stations is straightforward. The total labor cost term is linearly proportional to the number of stations on the line; the proportionality constant is $C \times L$. On the other hand, the total expected incompletion cost term is a monotonically nonincreasing function of K. As K increases, the total expected incompletion cost term decreases to an asymtote. Note that even with the maximum number of stations (K = N), there may remain a positive total expected incompletion cost; this quantity constitutes the asymtote.

The cost factors of the objective function can now be generalized to represent the total system cost function of a given allocation of tasks to stations. To that end, let there be K stations. The total labor cost of a station is $C \times L$. Then, the objective function of the model for a given allocation of tasks to K stations can be expressed as follows:

$$\text{Min } Z = C.K.L + \sum_{i=1}^{N} \left[\beta_i \left[IC_i + \sum_{k \in A_i} IC_k \right] - \sum_{j=1}^{fs_i} SB_i^j \right]$$

The optimal value of Z can be obtained by varying K and the allocations of tasks to these stations, such that

1. All tasks are allocated to stations,

2. No task is allocated more than once,

3. If task x precedes task y on the precedence diagram, then y is not allocated to a station that precedes the one to which x is assigned.

In the next chapter, the dynamic programming formulation of the problem is developed. The formulation guarantees the optimal solution if carried to completion, though the storage and computational requirements of the formulation grow very rapidly. An implementation of the formulation is also discussed.

4.0 Development Of A Methodology Based On Dynamic Programming

In this chapter, we present the dynamic programming formulation of the problem, and the implementation of the formulation is discussed. The dynamic programming approach to problem solving is discussed in Section 4.1. The dynamic programming formulation of the problem is developed in Section 4.2. An example is next solved to illustrate the formulation of the problem in Section 4.3. In Section 4.4, the implementation of the formulation is presented and the bounding strategy which reduces the storage and computational requirements of the formulation is developed. The computer implementation of the procedure is also discussed. Finally, in Section 4.5, computational experience on the bounding strategy is reported.

4.1 The Dynamic Programming Approach To Problem Solving

Optimization can be described as a process of finding the best solution from a set of alternatives for a problem which can be formulated quantitatively. The problem should be formulated in a form which is convenient for analysis. A conventional approach for the formulation of a problem is to construct a mathematical model that represents the essence of the problem. A lot of caution should be taken to ensure that the model is a valid representation of the problem. The basic components of a mathematical optimization model are as follows:

1. **Variables** are the quantities which can be manipulated to achieve some desired objective or performance measure.

2. The **objective function** is a measure of the performance or the value or utility which is associated with a particular combination of the variables. Constructing the objective function is a crucial step in formulating a mathematical model.

3. **Feasibility conditions** or **constraints** are the equations or inequalities that the variables must satisfy.

Dynamic programming is a sequential decision process that can be used to solve certain kinds of optimization problems. It is an approach to problem solving. It is a way of looking at a problem which may contain a large number of interrelated decision variables and determining the combination of decisions that optimizes overall effectiveness. By this approach, a n-variable problem is decomposed into n single variable problems. This decomposition reduces the computational effort. Solving n smaller problems requires a computational effort proportional to n, on the other hand, solving one larger problem with n variables usually requires a computational effort which is roughly proportional to z^n, where z is some constant [19]. A sufficient condition for a problem to be solved by the dynamic programming approach is the separability and monotonicity of the objective function.

Generally speaking, there is no standard way of formulating the dynamic programming approach. In other words, dynamic programming is not a well-defined procedure in the sense that Dantzig's simplex algorithm is a well-defined set of rules for solving a linear programming problem. The features of the problem need to be appropriately defined so that the resulting model is computationally effective. Typically a problem is decomposed as shown in Figure 7.

The problem is divided into stages, with a decision made at each stage. Each stage has a number of states, s_n, associated with it. The effect of the decision, d_n, made at each stage is to transform the current state, s_n, into a state associated with the next stage. In other words, $s_{n+1} = t_n(s_n, d_n)$. The principle that enables to carry out the transformation is known as the principle of optimality and is as follows: An optimal policy has the property that whatever the initial state and the initial decision are, the remaining decisions must constitute an optimal policy with respect to the state which results from the initial decision. In other words, every optimal policy consists only of optimal subpolicies. The solution starts by finding the optimal policy for each state of the final stage. A recursive relationship identifies the optimal policy for each state at stage n, given that the optimal policy for each state at stage n + 1 is available. Using the recursive relationship, the solution is found by moving backward stage by stage. At each stage, the optimal policy, decision, for each state is found.

Dynamic programming approaches can be classified as deterministic and probabilistic. In a deterministic approach, states at the next stage are completely determined by the states and policy decisions at the current stage. In the probabilistic case, there is a probability distribution for what the next state will be. Another classification relies on the direction of the recursion. In a forward-recursion dynamic programming problem, the states of stage n are identified from the states of stage n + 1. The opposite direction is the flow in a backward-recursion problem.

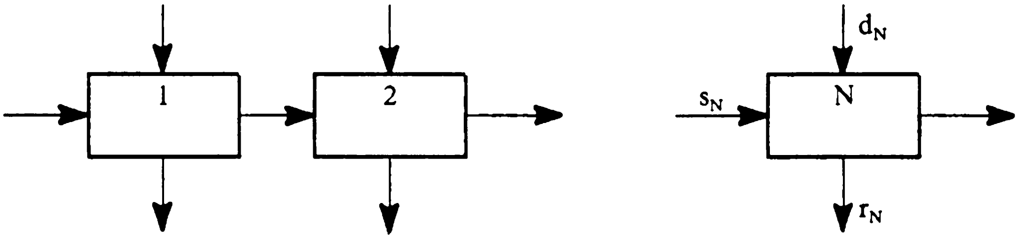


Figure 7. Decomposition of a dynamic programming formulation

4.2 Dynamic Programming Formulation

The features of the dynamic programming formulation of the single-model, stochastic assembly line balancing problem are as follows: The stages of the formulation are the stations on the line. The first stage corresponds to the first station, the second stage to the second station, and so on. Number of stations is a variable in the formulation. An upper bound on the number of stages is the number of tasks in the problem, since each station should at least accommodate a task. The state variable at stage n, s_n represents the set of tasks available for assignment at that stage. Thus, s_1 is the set of all the tasks in the problem. The decision variable at stage n, $x_n \in X_n$ represents the sequence of tasks to be assigned to station n, where X_n is the set of all possible sequences of tasks that can be assigned at stage n given s_{n+1} . The return function at stage n, $r_n(x_n, s_{n+1})$ is the total expected cost corresponding to decision variable x_n , and state variable s_{n+1} . In determining the decision variables, the precedence constraints are the only restrictions considered. The basic structure of the dynamic programming formulation is shown in Figure 8.

The return function, $r_n(x_n, s_{n+1})$ is similar to the objective function developed in Chapter 3 and can be expressed as follows:

$$r_n(x_n, s_{n+1}) = C\,L + \sum_{i \in x_n} \left[\beta_i \left[IC_i + \sum_{k \in A_i} IC_k \right] - \sum_{j=1}^{fs_i} SB_i^j \right]$$

Note that the return function represents the labor cost of a station and the expected incompletion cost of the tasks in the decision variable. If $f_n^*(s_{n+1})$ represents the cost of assigning the tasks in the set $\{s_1 - s_{n+1}\}$ to stages 1 through n, then the recursive relationship can be represented as:

$$f_n^*(s_{n+1}) = \min_{x_n \in X_n} \left\{ r_n(x_n, s_{n+1}) + f_{n-1}^*(s_n) \right\} \qquad \text{for } n = 1,\ldots,N$$

where $s_n = s_{n+1} + x_n$ and $f_0^*(.) = 0$ and $s_{N+1} = \emptyset$.

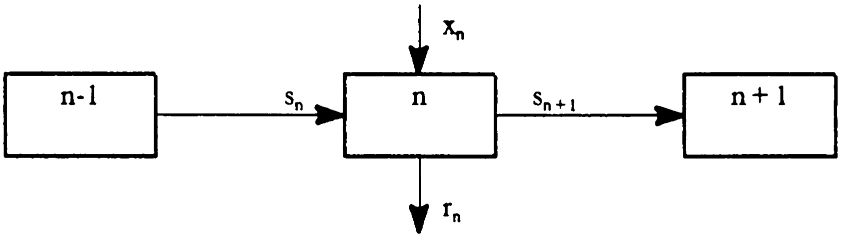


Figure 8. Basic structure of the dynamic programming formulation

One of the state variables is the null state for each stage except the first one; the null state indicates the assignment of all the tasks. The associated f_n^* function to the null state, $f_n^*(s_{n+1} = \emptyset)$ gives the optimal solution of the problem with n stations. Decisions made at stage n transforms s_n into the state variables of stage n + 1. Corresponding to each decision variable, x_n and state variables of stage n + 1, the return function $r_n(x_n, s_{n+1})$ is computed. Note that different decision variables can result in identical state variables for the next stage. Thus, a search to find the decision variable with the least return function is made for each state variable. The features of the dynamic programming formulation will be further clarified with the example problem solved in the following section.

4.3 *An Example*

An example problem with four tasks will be solved to clarify the features of the dynamic programming formulation of the problem. The parameters of the example problem are given in Table 4. Let L = 6.00 \$/hour and C = 10 minutes. The basic structure of the problem is depicted in Figure 9.

The recursive relationship relating the f_1^*, f_2^*, f_3^* and f_4^* functions is as follows:

$$f_n^*(s_{n+1}) = \min_{x_n \in X_n} \left\{ r_n(s_{n+1}, x_n) + f_{n-1}^*(s_n) \right\} \qquad \text{for } n = 1,\ldots,4$$

where $f_0^*(.) = 0$ and $s_5 = \emptyset$ and $s_n = s_{n+1} + x_n$ for n = 1,...,4 .

Table 4. Example problem parameters

Task (i)	Mean (μ_i) †	Var.(σ_i^2) ‡	Incompletion cost (IC_i) *	Cumulative incompletion cost (CIC_i) *	Immediate followers
1	4.0	0.8	2.0	10.0	2,3
2	2.0	0.4	1.0	4.0	4
3	8.0	1.6	4.0	7.0	4
4	6.0	1.0	3.0	3.0	None

† Mean values are in minutes
‡ Variance values are in (minutes)^2
* Cost values are in $

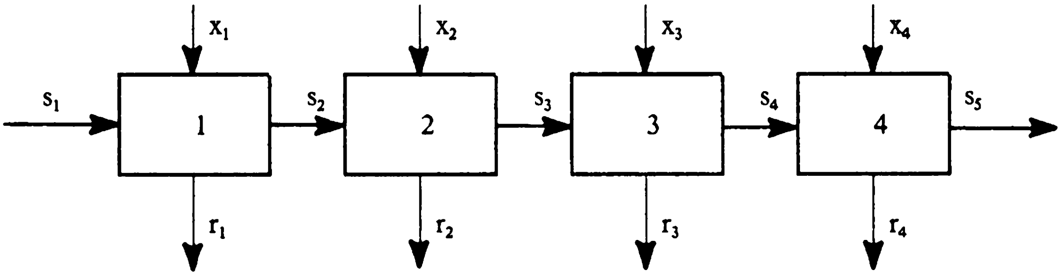


Figure 9. Basic structure of the example problem

The resulting dynamic programming calculations are given below:

Stage 1:

s_2	x_1	$f_1(s_2)$
{4}	{1,2,3}	7.941
	{1,3,2}	8.572
{2,4}	{1,3}	7.311
{3,4}	{1,2}	1.000
{2,3,4}	{1}	1.000
$\emptyset$	{1,2,3,4}	7.966
	{1,3,2,4}	8.597

Summary of Stage 1 calculations:

s_2	x_1	$f_1^*(s_2)$
{4}	{1,2,3}	7.941
{2,4}	{1,3}	7.311
{3,4}	{1,2}	1.000
{2,3,4}	{1}	1.000
$\emptyset$	{1,2,3,4}	7.966

Stage 2:

s_3	x_2	s_2	$f_2(s_3)$
{4}	{3}	{3,4}	2.400
	{2}	{2,4}	8.311
	{2,3}	{2,3,4}	5.500
	{3,2}		4.228
{2,4}	{3}	{2,3,4}	2.400
{3,4}	{2}	{2,3,4}	2.000
$\emptyset$	{4}	{4}	8.941
	{3,4}	{3,4}	5.209
	{2,4}	{2,4}	8.482

Summary of stage 2 calculations:

s_3	x_2	$f_2^*(s_3)$
{4}	{3}	2.400
{2,4}	{3}	2.400
{3,4}	{2}	2.000
$\emptyset$	{3,4}	5.209

Stage 3:

s_4	x_3	s_3	$f_3(s_4)$

{4}	{2}
	{3}
		{2,4}
		{3,4}
			3.400
			3.400

$\emptyset$	{4}
	{2,4}
	{3,4}
		{4}
		{2,4}
		{3,4}
			3.400
			3.571
			6.209

Summary of stage 3 calculations:

s_4	x_3	$f_3^*(s_4)$
{4}	{2} or {3}	3.400
$\emptyset$	{4}	3.400

Stage 4

s_5	x_4	s_4	$f_4(s_5)$
$\emptyset$	{4}	{4}	4.400

Optimal design with 4 stations:

1	3 or 2	2 or 3	4	Cost: 4.400 $/unit

Optimal design with 3 stations:

1,2	3	4	Cost: 3.400 $/unit

Optimal design with 2 stations:

1,2	3,4	Cost: 5.209 $/unit

Optimal design with 1 station:

1,2,3,4	Cost: 7.966 $/unit

As it is seen from the above calculations, the optimal design has 3 stations with a total system cost of 3.400 $/unit.

4.4 Implementation Of The Formulation

The dynamic programming formulation of the problem given in Section 4.2 would only obtain the solutions of problems of limited size, because of the excessive number of state and decision variables generated at each stage. Although several decision variables will be identical for different state variables and the precedence constraints prevent several decision variables from being generated, the number of state and decision variables would still be too large. The total number of state and decision variables generated by the dynamic programming procedure grows exponentially with an increase in the number of tasks. Thus, we need a procedure to prune the decision variables that are not expected to lead to the optimal solution. A sufficient number of decision variables at each stage should be pruned, so that the problem could be solved on the computer. On the other hand, the pruning of the decision variables should not result in a design with an operating cost much higher than the optimal design cost.

We will call the strategy that prunes some of the decision variables at each stage "bounding strategy". The bounding strategy imposes an upper bound on the incompletion probability of the decision variables. In other words, decision variables that have incompletion probabilities larger than a bound provided by the user are pruned. If α denotes this bound, then the decision variable, x_n is pruned if:

$$1 - \Phi\left(\frac{C - \sum_{i \in x_n} \mu_i}{\sqrt{\sum_{i \in x_n} \sigma_i^2}}\right) > \alpha$$

Theoretically, the range of α is between zero and one. The value of one corresponds to the generation of all possible decision variables at each stage. For practical reasons, it can be assumed that $\Phi(x) = 1 - \Phi(-x) = 0.0$ for $x \leq -3.0$. Therefore, a decision variable is assumed to be incomplete with certainty if the following condition is met:

$$\frac{C - \sum_{i \in A} \mu_i}{\sqrt{\sum_{i \in A} \sigma_i^2}} \leq -3.0$$

As α is decreased, the decision variables that have incompletion probabilities greater than α are discarded; this process decreases the computational and storage requirements of the dynamic programming formulation. However, the probability of missing the optimal design increases as α is decreased. The following Theorem determines a lower bound on α .

Theorem 4.1 $\underset{i=1,\ldots,N}{\mathrm{Max}} \left\{ 1 - \Phi(\frac{C - \mu_i}{\sigma_i}) \right\}$ constitutes a lower bound on α.

Proof. Note that $1 - \Phi(\frac{C - \mu_i}{\sigma_i})$ is the incompletion probability of task i, when assigned to a station by itself. For $\alpha < \underset{i=1,\ldots,N}{\mathrm{Max}}\{1 - \Phi(\frac{C - \mu_i}{\sigma_i})\}$, the procedure would not even consider inclusion of the task which determines $\underset{i=1,\ldots,N}{\mathrm{Max}}\{1 - \Phi(\frac{C - \mu_i}{\sigma_i})\}$, thereby violating the fact that all tasks must be performed.♦

Note that if $\mu_i \leq C$ for all i, then $\underset{i=1,\ldots,N}{\mathrm{Max}}\{1 - \Phi(\frac{C - \mu_i}{\sigma_i})\} \leq 0.5$. The above bounding strategy with α set to its lower bound determined by Theorem 4.1 enables the dynamic programming procedure described in Section 4.2 to solve problems of larger sizes that would require excessive storage and computation otherwise. On the other hand, the solution is no longer the optimum one. The total system cost of the solution found by utilizing the bounding strategy constitutes an upper bound on the total system cost of the optimal solution. However, this upper bound provides an excellent starting solution for the improvement procedure discussed in Chapter 5.

4.4.1 Computer Implementation Of The Dynamic Programming Procedure

The listing of the computer program written for the dynamic programming procedure is given in Appendix A. The program is written in FORTRAN and can handle problems of up to 50 tasks.

The cumulative number of decision and state variables over the stages are both limited to 15,000. N, L, C and α values should be provided to the program. μ_i, σ_i^2, IC_i , identities of the immediate preceding and following tasks should also be provided for all i. The program generates the minimum cost designs for number of stations starting at 1. For $\alpha < 1$, it is possible that a feasible solution cannot be obtained for a given number of stations. This is due to the fact that, if the number of stations is too small, then it may be impossible to meet the conditions that all the tasks are assigned to stations and the incompletion probability of each decision variable is less than α. On the other hand, a feasible solution is always obtained for a number of stations equal to or less than N, since α should be larger than the value of the lower bound determined by Theorem 4.1. The program reports such cases and the number of stations is increased until feasible solutions are obtained. The program also reports the CPU time spent after the initialization and each stage and the number of decision and state variables of each stage.

The main steps of the computer program are outlined below. Let NSTAGE denote the stage whose decision variables are currently generated.

Step 1. Read the data.

Step 2. Set the state variable of stage 1 to all the tasks in the problem.

Step 3. Set NSTAGE equal to 1.

Step 4. Generate the decision variables of stage NSTAGE. Compute the incompletion probability of each decision variable and disregard the ones with incompletion probabilities larger than α.

Step 5. Compute the return function of each decision variable generated and not disregarded in Step 4.

Step 6. Generate the state variables of stage NSTAGE+1. For each state variable of stage NSTAGE+1, find the decision variable of stage NSTAGE that yields the minimum total expected cost.

Step 7. If one of the state variables of stage NSTAGE+1 is a null set, then go to Step 9. Otherwise, go to Step 8.

Step 8. Increment NSTAGE by one. Go to Step 4.

Step 9. Find the solution corresponding to the null state variable of NSTAGE+1. If NSTAGE is equal to N, go to Step 10. Otherwise, go to Step 8.

Step 10. Find the least cost solution among the solutions corresponding to the null state variables of the stages. This solution constitutes the solution of the problem.

Step 11. Stop.

Note that in Step 7, the null state variable indicates the assignment of all the tasks up to that stage. Thus, a solution is obtained with NSTAGE stations. In Step 10, the least cost solution among the solutions with different number of stations is searched.

4.5 Computational Experience

To investigate the performance of the bounding strategy, several randomly generated problems with known optimum solutions were solved with an α value of 0.5. The optimal solutions were obtained using $\alpha = 1$. In the experimentation, two sets of problems with F-ratios 0.00 and 0.42 were created. Ten and 15-task problems were solved for the first set, 11-task problems were solved for the second set. In each category, 5 problems were solved. Computational and storage requirements restricted the attainment of the optimal solutions of problems with higher number of tasks and F-ratios. Cycle time was computed as $C \sim U[10;100]$. Task performance time parameters were computed as follows: $\mu_i \sim U[0;C]$, $\sigma_i = RAN_1 \mu_i$ and $IC_i = RAN_2 \mu_i$ for $i = 1,...,N$, where $RAN_1 \sim U[0.04 ; 0.06]$ and $RAN_2 \sim U[L ; 2L]$ and $L = 3.00$ \$/hour. OPT^* denotes the optimal solution, whereas $OPT(\alpha = 0.5)$ denotes the solution obtained using the bounding strategy with an α value of 0.5. The ratios of the solutions obtained with α set to 0.5 to the optimal ones are depicted in Table 5. The average of the ratio values is 1.12, and 90% confidence interval limits on the ratio values are 1.06 and 1.18, respectively. It is noted that the solutions obtained by setting α to 0.5 required negligible CPU time.

4.6 Conclusions

In this chapter, we formulated the stochastic, single-model assembly line balancing problem as a dynamic programming problem. Storage and computational requirements of the formulation grow very rapidly as the problem size increases; problems of even moderate sizes could not be solved due to the excessive storage and computational requirements. A bounding strategy is developed, so the storage and computational requirements of the formulation are reduced drastically. With the bounding strategy, problems of larger sizes could be solved, but the solutions are no longer the optimal ones. To investigate the effectiveness of the bounding strategy, an experimentation was made. The results indicate that the bounding strategy is very effective in reducing the storage and computational requirements of the formulation. Thus, these solutions constitute good initial solutions for the improvement procedure that will be developed in Chapter 5.

In the next chapter, the approximation procedure will be presented. This procedure divides the problem into subproblems and the improvement procedure is applied to each subproblem in order

Table 5. Comparison of the dynamic programming procedure solutions with an α of 0.5 to the optimal ones

F-ratio	Number of tasks	OPT(α=0.5)	OPT^*	$\frac{OPT(\alpha = 0.5)}{OPT^*}$
0.000	10	30.557	27.318	1.119
0.000	10	21.745	19.207	1.132
0.000	10	25.950	25.950	1.000
0.000	10	30.545	26.807	1.139
0.000	10	8.452	6.786	1.246
0.000	15	10.546	8.727	1.208
0.000	15	16.853	13.586	1.241
0.000	15	22.481	16.695	1.347
0.000	15	35.163	34.947	1.006
0.000	15	42.263	40.353	1.047
0.418	11	9.948	9.948	1.000
0.418	11	22.282	20.251	1.100
0.418	11	28.745	25.500	1.127
0.418	11	22.007	22.007	1.000
0.418	11	21.441	19.102	1.123

to improve the initial solution provided by the dynamic programming procedure with the bounding strategy.

5.0 Development Of A Methodology Based On The Approximation Procedure

In this chapter, we develop the approximation procedure for the single-model, stochastic assembly line balancing problem. The procedure divides the problem into subproblems. An initial solution is then generated for each subproblem, and it is further improved using a branch-and-bound type of procedure called the improvement procedure. The initial solution of each subproblem is obtained using the dynamic programming procedure with the bounding strategy as described in Chapter 4. This solution also acts as an upper bound of the improvement procedure. The improvement procedure either improves the initial solution of a subproblem or determines that it is very close to the optimal one. The improvement procedure is analogous to the branch-and-bound technique in that it considers all possible assignments of tasks to stations. However, it differs from the branch-and-bound technique due to the fact that an approximate solution of the remaining tasks (corresponding to a node) is computed instead of a lower bound. It is this solution that is used for pruning nodes. Consequently, the solution obtained by this scheme need not be the optimal solution; hence the name approximation procedure. However, if the approximate solution at every node is ε-optimal, we show that the final solution is also ε-optimal. The approximate solution of the partial assembly line balancing problem is obtained with the precedence constraints relaxed at every node of the tree. In the absence of precedence, this partial problem at every node is a M parallel machines scheduling problem. Thus, a heuristic procedure is developed for this relaxed problem that constructs a schedule on M parallel machines from a single-machine sequence. This procedure will be called "M-machine scheduling procedure". Some dominance properties are developed and implemented in this branch-and-bound type of procedure. One of the dominance properties requires the resequencing of the tasks assigned to the nodes of the enumeration tree. Thus, a procedure to sequence tasks on a single machine is developed. The single-machine sequence of the M-machine scheduling procedure is also constructed with this sequencing procedure.

An outline of the approximation procedure is given in the next section.

5.1 Outline Of The Approximation Procedure

The bounding strategy of the dynamic programming procedure presented in the previous chapter is quite effective in reducing the storage and computational requirements of the problem. However, as the problem size increases, the rapidly increasing storage and computational requirements of the problem exceed the computer capacity. The approximation procedure that will be presented in this section overcomes the rapid increase of the problem requirements by dividing the problem into subproblems.

In order to divide the problem into subproblems, the tasks are labelled as follows: First, the tasks which do not follow any other tasks are labelled as 1. Let h_i denote the label of task i. Then, $h_i = \underset{j \in Q_i}{\text{Max}} \{ h_j \} + 1$, where Q_i is the set of tasks immediately predecessing task i. After determining the labels of all the tasks, the tasks are numbered as follows: Let c_k be the number of tasks with label k. The tasks with label k (k > 1) are numbered in the increasing order starting with $\sum_{j=1}^{k-1} c_j + 1$. For k = 1, the tasks are numbered in the increasing order starting with 1. Numbering of the tasks with the same label is arbitrary; in other words, the tasks with the same label are numbered starting with any one of the tasks and going to the next one until all the tasks are numbered.

After numbering the tasks as described above, the approximation procedure is applied in accordance with the following main steps. Let N_{sp} denote the maximum number of tasks allowed in a subproblem.

Step 1. (Decomposition of the problem). Divide the problem into subproblems of N_{sp} or less tasks using the labelling scheme described above.

Step 2. (Determination of an initial solution of each subproblem). Obtain an initial solution to each subproblem using the dynamic programming procedure with α set to 0.5.

Step 3. (Improvement of the initial solution of each subproblem). Apply the improvement procedure to each subproblem in order to improve the initial solutions found in Step 2.

Step 4. (Determination of the final solution). Combine the solutions of the subproblems generated in Step 3 to obtain the final solution of the problem.

Steps 1 and 4 of the approximation procedure are presented in Section 5.2. Step 2 of the procedure can also be executed by techniques other than the dynamic programming procedure. It should be noted that a tight initial solution value can significantly reduce the size of the branch-and-bound type of tree generated during the improvement procedure of Step 3. The improvement procedure is discussed in Section 5.3.

5.2 Decomposition Of The Problem

The assembly line balancing problem can be decomposed into subproblems in several ways. Basically there are three decomposition methods; these are serial, parallel and any combination of serial and parallel decompositions. In serial decomposition, the tasks with labels $1,...,d_1$ belong to the first subproblem, the tasks with labels $d_1 + 1,...,d_1 + 1 + d_2$ belong to the second subproblem, and so on. Note that the labels of the tasks are analogous to the columns at which the tasks are located on the precedence diagram. Thus, serial decomposition can be interpreted as follows: The tasks in the first d_1 columns constitute the first subproblem, the tasks in the next d_2 columns constitute the second subproblem, and so on. In parallel decomposition, the tasks in the first d_1 rows of the precedence diagram constitute the first subproblem, the tasks in the next d_2 rows constitute the second subproblem, and so on. Finally, the serial and parallel decompositions can be combined to generate several other ways of decomposing the problem into subproblems.

As an illustration of these decomposition schemes, consider the example problem solved in Section 4.3. The labels of the tasks 1, 2, 3 and 4 are 1, 2, 2 and 3, respectively. Note that there are three columns on the precedence diagram of the problem. All possible ways of parallel decomposition of the problem are depicted in Figure 10. There are three ways of decomposing the problem serially and they are depicted in Figure 11. Finally, Figure 12 depicts the two ways the problem can be decomposed with the combinations of serial and parallel decompositions.

There are several factors that should be considered to determine which method of decomposition is used. These factors include the quality of the solution to the original problem, the feasibility of the final solution, and the convenience of obtaining the final solution.

Step 4 of the approximation procedure combines the solutions of the subproblems to obtain the final solution of the problem. In other words, say, the solution of the second subproblem is appended to the solution of the first subproblem, the solution of the third subproblem is appended

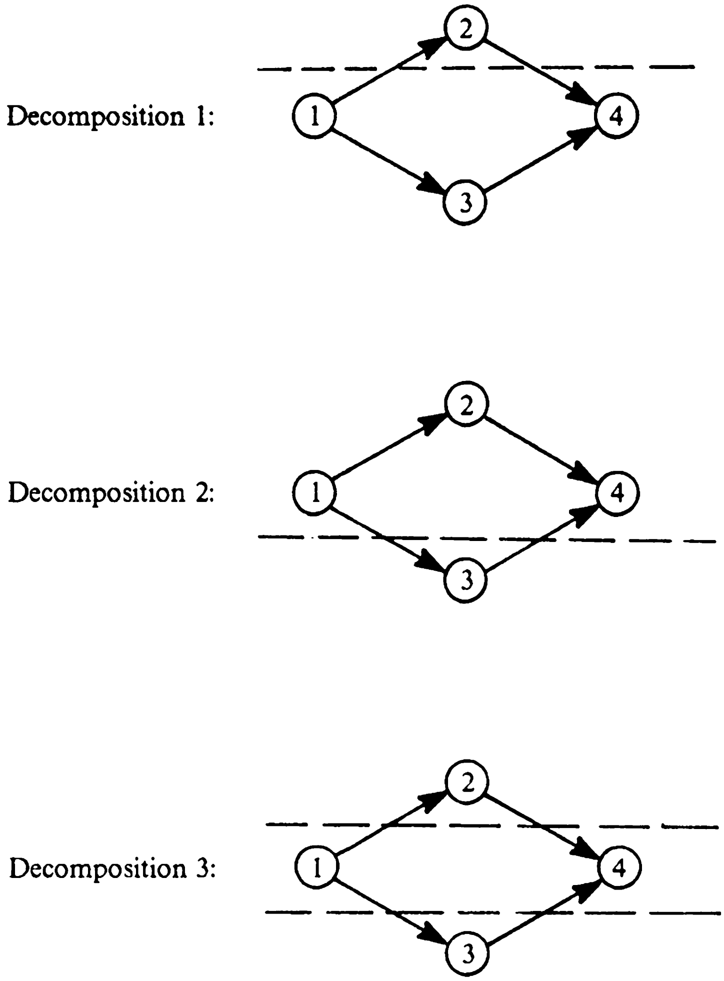
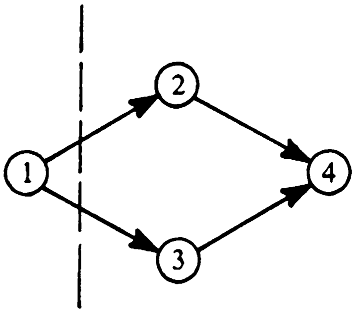
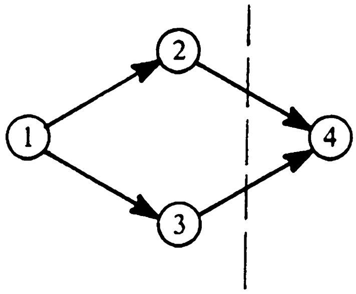


Figure 10. Parallel decomposition of the example problem of Section 4.3

Decomposition 1:



Decomposition 2:



Decomposition 3:

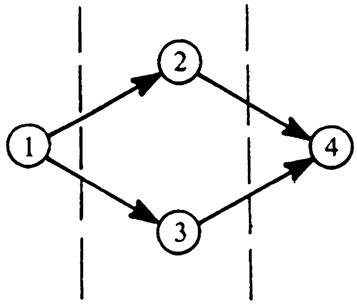


Figure 11. Serial decomposition of the example problem of Section 4.3

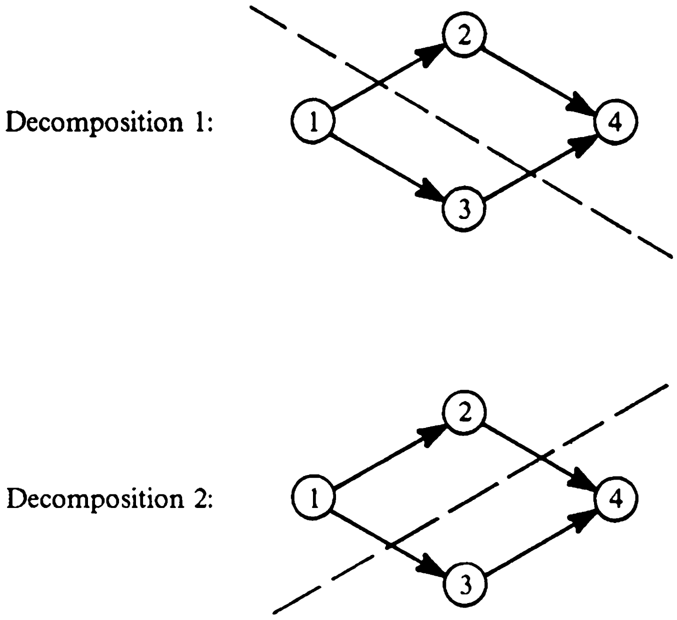


Figure 12. Decomposition of the example problem of Section 4.3 by combinations of serial and parallel decompositions

to the solutions of the first and second subproblems, and so on. The final solution of the problem should not violate precedence among tasks. Thus, if task i is in subproblem k ($k > 1$), then none of the tasks in set A_i should be in subproblems 1,...,k-1. In fact, after appending the solutions of the subproblems as described above, we can change the positions of the tasks that violate the precedence constraints. Thus, we do not need the condition above, since any solution can be turned into a feasible one. On the other hand, the computational effort for changing the positions of the tasks can be quite large and the solution value can deviate from the optimal one significantly. Therefore, we would like to append the solutions of the subproblems such that it does not require changing the positions of any tasks in the final solution to satisfy the precedence relations. If the problem is decomposed serially, then this condition is always satisfied. That is, if the first subproblem has the tasks in the first d_1 columns, the second subproblem in the next d_2 , and so on, the final solution of the problem can be simply obtained by appending the solutions of these subproblems. Note that parallel decomposition never satisfies this condition except for the rare case when the F-ratio of the precedence diagram of the problem is 0.0 (then, we cannot decompose the diagram by parallel decomposition, since the precedence diagram has just one row).

Serial decomposition is not the only method that allows Step 4 of the approximation procedure to obtain feasible solutions for the problem. Some combinations of serial and parallel decompositions can also lead to feasible final solutions. Note that the final solution of the problem after appending the subproblem solutions is desired to be as close to the optimal solution as possible. It is more likely that the final solution is closer to the optimal one when the problem is decomposed serially compared to any other method of decomposition, because the number of feasible sequences that can be generated by the tasks in the subproblems in that case are larger than when decomposed another way. Note also that the F-ratios of the subproblems are larger with serial decomposition compared to any other method.

Another factor in decomposing the problem into subproblems is the convenience of implementing the procedure. Serial decomposition is the most attractive method in this respect as no changes are required in Step 4 to obtain the final solution.

The final solutions corresponding to the decompositions depicted in Figures 10, 11 and 12 are shown below.

Final solutions with parallel decomposition:

Decomposition 1: Tasks in station 1 : 2
Tasks in station 2 : 1
Tasks in station 3 : 3
Tasks in station 4 : 4
Cost = 4.400 $/unit

Decomposition 2: Tasks in station 1 : 1,2
Tasks in station 2 : 4
Tasks in station 3 : 3
Cost = 3.400 $/unit

Decomposition 3: Tasks in station 1 : 2
Tasks in station 2 : 1
Tasks in station 3 : 4
Tasks in station 3 : 3
Cost = 4.400 $/unit

As it is seen, all of the parallel decompositions above lead to infeasible solutions for the original problem. In Decomposition 1, if the task of station 1 is exchanged with the task of station 2, then the solution becomes feasible. The same exchange is necessary between the tasks of stations 2 and 3 in Decomposition 2. In Decomposition 3, the tasks of stations 1 and 2 and the tasks of stations 3 and 4 should be exchanged. Note that such exchanges can be quite time consuming and the solution value can deviate from the optimal one significantly for problems of larger sizes.

Final solutions with serial decomposition:

Decomposition 1: Tasks in station 1 : 1
Tasks in station 2 : 2
Tasks in station 3 : 3,4
Cost = 6.209 $/unit

Decomposition 2: Tasks in station 1 : 1,2
Tasks in station 2 : 3
Tasks in station 3 : 4
Cost = 3.400 $/unit

Decomposition 3: Tasks in station 1 : 1
Tasks in station 2 : 2
Tasks in station 3 : 3
Tasks in station 4 : 4
Cost = 4.400 $/unit

Note that the optimal solution, as shown in Section 4.3, is obtained with Decomposition 2 above.

Final solutions with the combinations of serial and parallel decomposition:

Decomposition 1: Tasks in station 1 : 1
Tasks in station 2 : 3
Tasks in station 3 : 2,4
Cost = 3.536 $/unit

Decomposition 2: Tasks in station 1 : 1,2
Tasks in station 2 : 3
Tasks in station 3 : 4
Cost = 3.400 $/unit

Decomposition 2 above also generates the optimal solution.

Decomposition of a problem into subproblems will be performed serially due to the reasons discussed above. The decomposition process can now be described as follows. Let d denote the column that contains the task with the task number N_{sp} . If column d does not contain any tasks with task numbers larger than N_{sp} , then the tasks in the first d columns constitute the first subproblem. Otherwise, the tasks in the first d-1 columns constitute the first subproblem. Note that each subproblem can have at most N_{sp} tasks. The tasks considered in a subproblem are deleted from the precedence diagram and the division process is reapplied until all the tasks in the problem are considered in one of the subproblems. Note that a problem can have more than N_{sp} tasks with the same label. For such rare cases, the set of tasks with the same label are divided into subsets of N_{sp} tasks, and each subset is regarded as another set with a different label.

After the division of the problem into subproblems, an initial solution is obtained for each subproblem using the dynamic programming procedure with the bounding strategy. The α value used in the dynamic programming procedure program could be increased to yield better results, since the sizes of the subproblems permit α to be quite close to unity. The improvement procedure is then applied to each subproblem; it either improves the initial solution or determines that it is quite close to the optimal one.

5.3 Improvement Procedure

In this section, we present the improvement procedure that is applied to the initial solution obtained by the dynamic programming procedure described in Chapter 4 with α set to 0.5. In the sequel, we first describe the improvement procedure and then a detailed discussion of various features of the procedure is presented.

5.3.1 Development Of The Improvement Procedure

The improvement procedure is analogous to the branch-and-bound technique in many respects. A tree is formed representing the station assignments, and the nodes that do not lead to the optimal solution are pruned. The main difference between this procedure and the branch-and-bound technique is in the evaluation of the nodes. Branch-and-bound technique requires a lower bound at each node, whereas the improvement procedure generates an approximate solution at each node which may not be a lower bound. Thus, the improvement procedure is not, in true sense of the word, the branch-and-bound technique.

The performance measure to be optimized is the total system cost of the assembly line as discussed in Chapter 3. A tree is formed with nodes representing station assignments of the tasks. Each level of the tree corresponds to a station; in other words, first level corresponds to the first station, second level to the second station, etc. Note that the tree can have at most N levels corresponding to N stations with a task in each station. The nodes of the next level are formed by considering the precedence constraints. Each node has an associated relaxed problem which is defined as follows: A relaxed problem corresponding to a node consists of the tasks that are not in the node or in the parent nodes. The precedence constraints among the tasks are relaxed. Let CN_i be the cost associated with the assignment of tasks to node i and n be the number of nodes generated in the tree. Let $TCN_i = TCN_{\zeta_i} + CN_i$, where ζ_i is the parent node of node i. TCN_i represents the cost associated with the assignment of the tasks to node i and all its parent nodes. Note that if node i has no parent node (e.g., the nodes in the first level), then $TCN_{\zeta_i} = 0.0$ and $TCN_i = CN_i$. An approximate cost is computed for each node of the tree. If APP_i denotes the approximate cost corresponding to node i, then APP_i is defined as follows:

$$APP_i = TCN_i + CRX_i$$

where CRX_i is the cost of the relaxed problem corresponding to node i.

In the branch-and-bound technique, CRX_i represents a lower bound on the contribution of the remaining tasks to the partial solution of node i. However, in our case, CRX_i need not be a lower bound, as it is a heuristic solution to the relaxed problem. CRX_i is used just like the lower bound value to fathom nodes. Thus, the improvement procedure is not guaranteed to result in the optimal solution. However, if CRX_i is an ε-optimal solution, then it can be shown to generate an ε-optimal solution of the original problem in the following sense. Let UB_{cur} denote the current upper bound. Since the optimal solution belongs to one of the branches of the tree, then in the worst case, node i containing the optimal solution is fathomed subject to:

$$APP_i = (1 + \varepsilon) \text{ Optimal solution } \geq UB_{cur}$$

$$\text{Optimal solution } \geq \frac{UB_{cur}}{(1 + \varepsilon)}$$

Thus, UB_{cur} will be at most within $(1 + \varepsilon)$ of the optimal solution.

The advantages of using an ε-optimal solution at a node, instead of a lower bound, are that (i) it is easy to obtain and (ii) it is close to the optimal solution value and hence is very effective in cutting down the size of the tree. Of course, the disadvantage is that it can guarantee only ε-optimal solutions. Thus, the smaller the value of ε, the better is this approximation procedure. The magnitude of ε depends on the performance of the M-machine scheduling procedure used to solve the relaxed problem at every node of the tree. An experimentation is conducted to investigate the magnitude of ε; it will be presented in Section 5.3.3.1.

During the generation of the enumeration tree, if the solution of the relaxed problem satisfies the precedence constraints, then the associated node yields a feasible solution. Such nodes are called feasible nodes and are fathomed. If the solution corresponding to the feasible node is smaller than UB_{cur}, then it becomes the current incumbent solution and the associated cost replaces UB_{cur}. Then, the unexplored nodes (unpruned nodes) with approximate costs larger than the new UB_{cur} are pruned. The procedure continues until no unexplored nodes are left in the tree.

If the node is not a feasible node, then the approximate cost of the node is computed and compared with UB_{cur}. If the approximate cost is greater than UB_{cur}, then the node is pruned, since that node is not expected to yield a solution better than the current incumbent solution.

In summary, there are two pruning tests for each node. Node i is pruned if one of the tests below is satisfied:

Pruning Test 1. $APP_i > UB_{cur}$.

Pruning Test 2. The solution of the associated relaxed problem satisfies the precedence constraints, thereby implying that node i is a feasible node.

If node i is pruned with Pruning Test 2 and the solution corresponding to the node is smaller than UB_{cur}, then the incumbent solution is replaced by the solution corresponding to node i and UB_{cur} is set equal to the solution corresponding to the node. Then, all the other unexplored nodes in the tree are checked with the new UB_{cur} and the qualified ones are pruned.

The main steps of the improvement procedure can be outlined as follows:

Step 1. Set UB_{cur} equal to the solution obtained using the dynamic programming procedure with α value at 0.5.

Step 2. Generate the nodes of level 1. For each node, compute CN and APP values. Check if the nodes could be pruned by the pruning tests and prune the ones that qualify. Reset UB_{cur} and change the incumbent solution if necessary.

Step 3. If there are no unexplored or unbranched nodes left, stop. Otherwise, go to Step 4.

Step 4. Select the node with the least approximate cost and branch from it to form the nodes of the next level with the node generation process described in Section 5.3.2.

Step 5. For each new node generated, compute TCN and APP values. Check if the node could be pruned by the pruning tests and prune if qualified. Reset UB_{cur} and change the incumbent solution if necessary. If a new incumbent solution is obtained, check all the unexplored nodes in the tree with the new UB_{cur} and prune the qualified ones. Go to Step 3.

If TCN found for a node in Step 5 is larger than UB_{cur}, then there is no need to solve the relaxed problem corresponding to the node. The node is pruned, since the approximate cost of the node will also be larger than UB_{cur}. Note that the same situation applies to Step 2; if CN of a node in Step 2 is larger than UB_{cur}, the node is pruned without solving the corresponding relaxed problem.

In the following sections, various features of this procedure are discussed in detail.

5.3.2 Branching Scheme

In this section the branching process of a node into nodes of the next level is described in detail. Several features of the scheme that contribute to the effectiveness of the improvement procedure are discussed.

The branching rule for selecting a node to partition the solution space is the best bound rule. The bound having the smallest approximate cost is selected to branch from, because this subset would seem to be the most promising one to contain the optimal solution.

The generation of the nodes of the next level from a node is as follows. The tasks that are available for assignment are placed in stage 1 and are considered marked. An immediate follower of a state S is defined as a task that is an immediate follower of at least one of the tasks in S and is not preceded by any tasks not in S. The unmarked immediate followers of a state are augmented to the current state to form the states of the next stage. The augmentation of states and corresponding unmarked immediate followers is done in stages. For any state S of stage k, the unmarked immediate followers are placed in a list F(S). Let H be a subset of F(S), then $S \cup H$ is a state for stage $k+1$. For each state of stage k, the unmarked immediate followers are found and placed as marked tasks for stage $k+1$. When all the tasks are marked or F(S) gets empty for the current stage, the generation procedure is complete. The states constitute the nodes of the next level corresponding to the node being branched from. The procedure generates all possible nodes corresponding to the parent node if carried to completion. This branching scheme is illustrated next using the example of Section 4.3. Suppose that we wish to branch from the initial node. The generation of the nodes is shown in Table 6. Initially, task 1 is the only available task for assignment and is placed in stage 1 and considered marked. The unmarked immediate followers of task 1 are tasks 2 and 3; they are placed in list F(S) corresponding to task 1. Note that tasks 2 and 3 are also placed in Stage 2 as marked tasks. Then, task 1 is augmented to all subsets of the list F(S) containing tasks 2 and 3 to form the states of Stage 2. For each state in Stage 2, the corresponding F(S) list is found. Note that for some states, this list is empty. The node generation process continues in this manner. In the example above, the procedure terminates in Stage 3, because F(S) is empty for all the states of that stage. As it is seen from Table 6, the initial node is branched into seven nodes.

The node generation process results in all possible nodes corresponding to a parent node if carried to completion. Next, we develop two dominance properties which help in significantly reducing the size of the tree and contribute in enhancing the effectiveness of the improvement procedure. Note that these properties apply to the descendent nodes of a parent node that are generated by the process described above. These dominance properties are as follows: Let the tasks in a node of stage $k+1$ of the node generation process be denoted by $S \cup H_j$, where S is a state of stage k and H_j is a subset of F(S).

Dominance Property 1. If TCN associated with the node $S \cup H_j$ is larger than UB_{cur}, then all the other nodes formed by $S \cup H_k$, where H_j is a subset of H_k, are pruned.

After generating each node, the cost associated with the assignment of the tasks to the node and all its parent nodes, TCN, is computed. The process of computing TCN for the node will be fur-

Table 6. Generation of nodes from the initial node of the example problem of Section 4.3

Stage	Marked tasks	State, S	Unmarked immediate followers, F(S)
1	1	1	2,3
2	2,3	1,2	
		1,3	
		1,2,3	4
		1,3,2	4
3	4	1,2,3,4	
		1,3,2,4	

ther discussed in Section 5.2.3. If TCN associated with the node $S \cup H_j$ is larger than UB_{cur}, the node is pruned. Furthermore, all the other nodes formed by $S \cup H_k$, where H_j is a subset of H_k are also pruned, since TCN associated with these nodes will also be larger than UB_{cur}. To illustrate this in the example above, suppose that the TCN associated with the node {1,3} is larger than UB_{cur}. Then, the nodes {1,2,3} and {1,3,2} will also have TCN higher than UB_{cur} and are not generated.

After the generation of each node, the total expected incompletion cost of the tasks in the node may be decreased by resequencing the tasks. Note that the resequencing process should not result in a sequence that violates the precedence constraints. When the tasks in a node are resequenced and TCN associated with the node is computed, Dominance Property 2 prunes the node without solving the corresponding relaxed problem if the node is dominated by one of the other descendent nodes.

Dominance Property 2. If TCN associated with a node is larger than that of any other descendent nodes generated previously with the same set of tasks, then the node is pruned. If the TCN is smaller than that of the previously generated node, then the previously generated node is pruned.

When a node is generated, it is resequenced with the single-machine sequencing procedure that will be described in Section 5.3.2.2. Since the single-machine sequencing procedure is a heuristic procedure, it is not guaranteed that this procedure generates the optimal sequence. But, it can potentially decrease the expected incompletion cost of the sequence. Note that if the original sequence corresponding to a node is the optimal one, then it remains intact in the resequencing process. After the resequencing process, TCN of the node is computed. When another descendent node of the parent node is generated with the same set of tasks sequenced in a different order than one of the previously generated nodes, the tasks are resequenced and the node with the higher TCN is pruned. Note that only one node among those containing the same set of tasks and corresponding to a parent node can remain unpruned, because the nodes with the same set of tasks have the same relaxed problems and if they lead to feasible solutions, the solution corresponding to the node with the larger TCN has a higher total system cost than the other solution.

Next, we describe the implementations of these Dominance Properties.

5.3.2.1 Implementation Of The Dominance Properties

When the node with the least approximate cost is selected by Step 4 of the improvement procedure, it is branched into a node of the next level with the node generation process described above. Tasks assigned to the node are resequenced and TCN of the node is computed. Then, the approximate cost of the node is computed with the M-machine scheduling procedure. The node is pruned if the approximate cost of the node is larger than UB_{cur} or if the node is a feasible one. The parent node is then branched into another descendent node; the tasks of this node are also resequenced and TCN of the node is computed. If the set of tasks of the new node are the same set of tasks as that of the previous node, then the node with the larger TCN is pruned, since it is dominated by the node with the smaller TCN. Note that these two nodes have the same relaxed problems and if these nodes lead to feasible solutions, then the node with the smaller TCN leads to the feasible solution with a smaller total system cost. Therefore, only one descendent node containing the same set of tasks corresponding to a parent node can remain unpruned; whenever a new descendent node is generated with the same set of tasks, Dominance Property 2 prunes one of the nodes. If the pruned node is the new node, then the relaxed problem corresponding to the node is not solved, either.

Dominance Property 2 contributes to the effectiveness of the improvement procedure significantly by eliminating the computations required for the solutions of the relaxed problems of several nodes. In addition, it reduces the size of the tree by pruning the nodes as described above. On the other hand, note that this property never prunes the node with tasks ordered in the optimal sequence. Whenever a node is generated with tasks ordered in the optimal sequence, it remains intact in the resequencing procedure. Thus, any node with the same set of tasks ordered in a different sequence has a higher TCN and is pruned. Note that if the resequencing procedure were guaranteed to generate the optimal sequence, then the nodes with the same set of tasks would have a unique sequence which is the optimal one. Since Dominance Property 2 never misses the optimal sequence of the tasks, the ε-optimality of the final solution is also guaranteed.

When Dominance Property 2 is applied to the nodes after the tasks of the nodes are resequenced, the same set of tasks can be resequenced several times in different nodes of the tree. This repetition can potentially be eliminated as follows. All the combinations of the tasks are formed. The sequences corresponding to each combination are examined to find the one with the least expected incompletion cost, and the optimal sequences are stored in a list. Note that these sequences should not violate the precedence constraints. Whenever a node is generated, the optimal sequence corresponding to the set of tasks of the node is found and replaced with the node without any resequencing process. No other node with the same set of tasks is generated from that parent node. Although this procedure seems computationally more attractive, the formation of the optimal sequences corresponding to each combination requires an enormous effort, because the number of combinations is quite large for problems of even small sizes. In addition, the list to store the optimal sequences requires immense storage. Therefore, the improvement procedure overcomes these restrictions by repeating the resequencing process for some nodes of the tree.

The repetition of the resequencing process for different nodes of the tree is partially eliminated by Dominance Property 1. If TCN associated with the node $S \cup H_j$ is larger than UB_{cur}, then all the other nodes formed by $S \cup H_k$, where H_j is a subset of H_k are pruned without resequencing the tasks in them. Since TCN of the node $S \cup H_j$ is computed after the tasks in the node are resequenced, the nodes formed by $S \cup H_k$ will also have TCN's larger than UB_{cur} after the resequencing process. Thus, these nodes are pruned without resequencing the tasks in them.

In the next section, the procedure to resequence the tasks of a node is developed.

5.3.2.2 Procedure To Sequence Tasks On A Single Machine

The problem of resequencing the tasks of a node can be viewed as the problem of sequencing N tasks on a single-machine with tasks having a common due date and stochastic processing times. Consequently, a task, if not completed within the due date, incurs a cost equivalent to its cumulative incompletion cost. The objective is to sequence tasks so that the expected incompletion cost is minimized. This problem is like a single-machine sequencing problem with a nonlinear loss function, however, the loss function here is defined as the expected incompletion cost. The sequencing procedure developed here will also be used to construct the initial single-machine sequence of the M-machine scheduling procedure used to solve the relaxed problems corresponding to the nodes of the enumeration tree of the improvement procedure. After the M-machine schedule is obtained for a relaxed problem, the tasks within each machine are also resequenced with this procedure.

In the sequel, we first review the related literature on the single-machine sequencing problem. Then, the notation and assumptions used in the development of the procedure are discussed. The development of the procedure is then presented.

Related Literature On The Single-Machine Sequencing Problem: Several studies have been reported in the literature for the single-machine problem with nonlinear loss functions. One of the earlier attempts for solving the problem was made by McNaughton [66], who described a procedure for finding the optimal schedule to the single-machine problem with linear loss functions and deterministic task processing times. Lawler [60] extended McNaughton's [66] study for nonlinear loss functions by using a dynamic programming approach. Lawler [60] also presented some linear programming formulations for the multiple-machine case with nonlinear loss functions and deterministic processing times. Schild and Fredman [89] developed criteria for quadratic loss functions to determine the relative order in which two tasks should appear in the optimal sequence. For general loss functions, the number of computations required by this algorithm grows exponentially with increase in N. Baker and Schrage [6,90] developed a dynamic programming approach for the single-machine problem with precedence constraints among the tasks in which the loss function is a function of the time at which tasks are started in the sequence. Their approach is developed with the aim of cutting down on the storage requirement of dynamic programming and consequently can be applied to problems with relatively large values of N. Steiner [95] presented an improved dynamic programming approach to the same problem by defining a compact labeling scheme and an efficient enumerative procedure for all the feasible subsets which are the state variables of the approach. Townsend [104] developed a branch-and-bound solution to the single-machine problem with quadratic loss function of task flowtimes. The procedure is not practical for large problems, and an approximate solution which requires generation of $\frac{1}{2}N(N + 1)$ nodes is recommended. Bagga and Kalra [5] further suggested a node elimination procedure for Townsend's [104] algorithm. Gupta and Sen [37] curtailed the enumeration tree of Townsend's [104] algorithm at the branching stage by recognizing certain conditions which give a priori precedence relations among some of the tasks in the optimal sequence. Regarding the consideration of stochastic procesing times of tasks, one of the earlier attempts was made by Banarjee [7] for a single-machine problem. Lately, considerable research has been reported in the area of stochastic scheduling. For a review, the reader is referred to papers by Pinedo [71] and Pinedo and Weiss [72].

Notation And Assumptions: Consider a single facility with N tasks waiting. Assume that the facility is free at the moment, and we wish to decide the sequence in which the tasks should be processed on that facility. The performance measure to be optimized is the expected incompletion cost.

Let

$f_i(s)$ = flowtime of task i in sequence $s \in S$, for $i = 1,\ldots,N$

The performance measure can be expressed as

$$\underset{s \in S}{\text{Min}} \sum_{i=1}^{N} IC_i \times \Pr\left\{ f_i(s) > C \right\}$$

where S is the set of all permutations of the N tasks.

Since the task performance times are independent and distributed normally with known means, then $f_i(s)$ is a normally distributed random variable for all i and s. Task performance time standard deviations are expected to be proportional with their means as, for example, a task with a large expected processing time contains a large number of elementary jobs, consequently resulting in a large standard deviation of the task. Accordingly, let $\sigma_i = a \times \mu_i$ for $i = 1,\ldots,N$, where a is a constant.

Consider an arbitrary sequence R in which a pair of adjacent tasks, i and j, with j following i, exists such that $IC_i \geq IC_j$. In the sequence R′, the tasks i and j are interchanged in sequence. The situation is depicted in Figure 13.

Let

$$p_i = 1 - \Phi\left[\frac{C - (\mu_Z + \mu_i)}{\sqrt{\sigma_Z^2 + \sigma_i^2}}\right] = \text{incompletion probability of task i in sequence R}$$

$$p'_j = 1 - \Phi\left[\frac{C - (\mu_Z + \mu_j)}{\sqrt{\sigma_Z^2 + \sigma_j^2}}\right] = \text{incompletion probability of task j in sequence } R'$$

p_Z = incompletion probability of the task preceding task i in sequence R or task j in sequence R′

$$p_Z = 1 - \Phi\left[\frac{C - \mu_Z}{\sigma_Z}\right]$$

p = the incompletion probability of task j in sequence R or task i in sequence R′

$$p = 1 - \Phi\left[\frac{C - (\mu_Z + \mu_i + \mu_j)}{\sqrt{\sigma_Z^2 + \sigma_i^2 + \sigma_j^2}}\right]$$

where μ_Z and σ_Z^2 are the sum of the means and variances of the tasks in Z, respectively. Also let Z_i denote the set of tasks preceding task i and Cost(R) be the expected incompletion cost of sequence R.

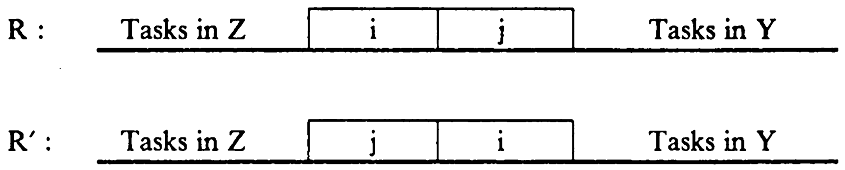


Figure 13. Relative orders of tasks i and j in sequences R and R′

In the next section the development of the sequencing rule is presented.

Development Of The Sequencing Rule: Let x denote the sum of the means of the tasks assigned to a station, then the incompletion probability $p(x) = 1 - \Phi(\frac{C - x}{a.x}) = \Phi(\frac{x - C}{a.x})$. For $x = C$, $p(x) = 0.5$, since $\Phi(0.0) = 0.5$, and p(x) approaches to one as x increases. Figure 14 depicts a portion of the incompletion probability function and shows the incompletion probabilities of two tasks, i and j, assigned to the same position in a sequence and the incompletion probability of the later task when they are assigned one after each other.

If certain conditions are met, we can determine the relative order of two adjacent tasks in order to minimize the total expected incompletion cost. Note that $\beta_i = p_i - p_Z$, $\beta'_i = p - p'_j$, $\beta_j = p - p_i$ and $\beta'_j = p'_j - p_Z$. Let CIC_i and CIC'_i be the cumulative incompletion cost of task i in sequence R and R′, respectively. The following Theorem states the sequencing rule for two adjacent tasks and the conditions that should be met.

Theorem 5.1. If $\beta_i CIC_i \le \beta'_i CIC'_i$ and $\beta_j CIC_j \le \beta'_j CIC'_j$, then $Cost(R) \le Cost(R')$ and if $\beta_i CIC_i \ge \beta'_i CIC'_i$ and $\beta_j CIC_j \ge \beta'_j CIC'_j$, then $Cost(R) \ge Cost(R')$.

Proof. $Cost(R) = \beta_i CIC_i + \beta_j CIC_j$ and $Cost(R') = \beta'_j CIC'_j + \beta'_i CIC'_i$
Therefore, $Cost(R) - Cost(R') = (\beta_i CIC_i + \beta_j CIC_j) - (\beta'_j CIC'_j + \beta'_i CIC'_i)$
Since $\beta_i CIC_i \le \beta'_i CIC'_i$ and $\beta_j CIC_j \le \beta'_j CIC'_j$, it implies that $Cost(R) \le Cost(R')$
The second part of the Theorem can be proved similarly. ♦

The above result implies that if two adjacent tasks, i and j, have expected processing times and cumulative incompletion costs as $\beta_i CIC_i \le \beta'_i CIC'_i$ and $\beta_j CIC_j \le \beta'_j CIC'_j$, then task j should follow task i in the sequence. If they are as $\beta_i CIC_i \ge \beta'_i CIC'_i$ and $\beta_j CIC_j \ge \beta'_j CIC'_j$, then task i should follow task j. Note that the above result applies to any adjacent pair of tasks irrespective of their positions in the sequence.

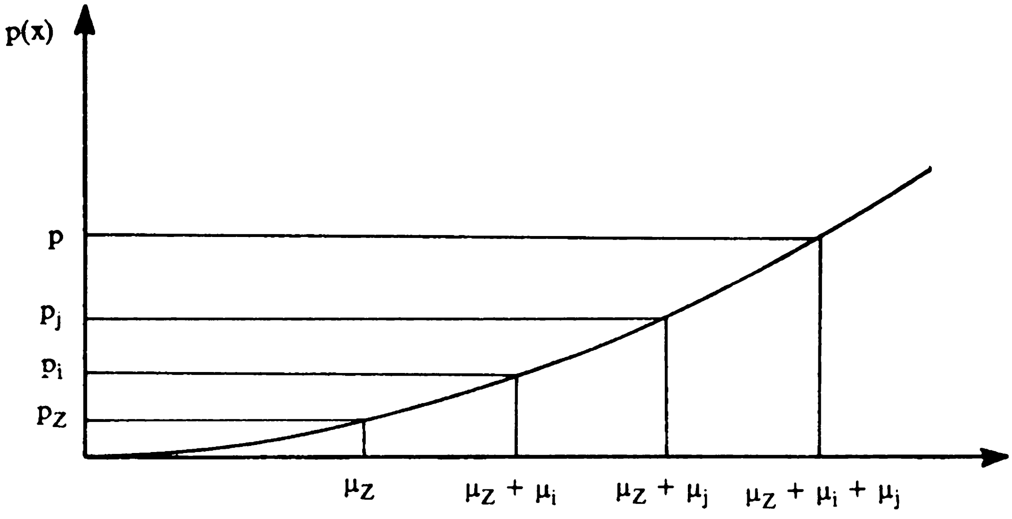


Figure 14. Incompletion probabilities of tasks i and j assigned to the same position in a sequence

The sequencing procedure first orders the tasks in the descending order of their cumulative incompletion costs with the precedence constraints being observed. If the order of the tasks violates the precedence constraints, then the positions of the tasks violating the constraints are changed accordingly, though the sequence of the tasks no longer remains to be the descending order of the cumulative incompletion costs of the tasks. Then, the conditions of the above Theorem are applied to each pair of adjacent tasks in the sequence to arrange them accordingly. If the conditions of the above Theorem are not met, then the expected incompletion cost of the sequence with task j following task i should be compared with that of the sequence with task i following task j. But, in certain regions of the sequence, the incompletion probabilities of the tasks i and j cannot be differentiated; they are both assumed to be either negligible or equal to unity due to the asymptotic shape of the Normal distribution. Assuming that $\sigma_i = a \times \mu_i$ for $i = 1,\ldots,N$ and $\Phi(x) = 1 - \Phi(-x) = 0.0$ for $x \leq -3.0$, we can determine the regions in which the incompletion probabilities of the tasks i and j are not differentiated as follows. Let E denote the value of μ_Z below which the incompletion probabilities of the tasks i and j are both negligible.

$$1 - \Phi\left(\frac{C - \mu_Z}{a\mu_Z}\right) = 1 - \Phi\left(\frac{C - E}{aE}\right) = 0.0$$

$$\frac{C - E}{aE} \geq 3.0$$

Then $E \leq \dfrac{C}{1 + 3a}$

And let F denote the value of μ_Z beyond which the incompletion probabilities of the tasks i and j are both assumed to be equal to unity. That is,

$$1 - \Phi\left(\frac{C - \mu_Z}{a\mu_Z}\right) = 1 - \Phi\left(\frac{C - F}{aF}\right) = 1.0$$

$$\frac{C - F}{aF} \leq -3.0$$

Then $F \geq \frac{C}{1 - 3a}$

If tasks i and j are in a region such that $E < \mu_Z < F$ and the conditions of Theorem 5.1 are not met, then the relative order of the tasks is determined by comparing Cost(R) with Cost(R′) . If Cost(R) < Cost(R′) , then task j follows task i; otherwise task i follows task j. In summary, there are three possible situations when the relative order of two adjacent tasks assigned to the same node of the tree formed by the improvement procedure is determined in order to minimize the total expected incompletion cost. The situations and the corresponding means of determining the relative order of the tasks in the node are summarized as follows:

Situation 1. The adjacent tasks meet the conditions of Theorem 5.1. The relative order of the tasks is determined according to Theorem 5.1.
Situation 2. The adjacent tasks do not meet the conditions of Theorem 5.1 and $E \leq \mu_Z \leq F$. The relative order of the tasks is determined by comparing the expected incompletion costs of the sequences corresponding to the two positions of the tasks.
Situation 3. The adjacent tasks do not meet the conditions of Theorem 5.1 and $\mu_Z < E$ or $\mu_Z > F$. The optimal relative order of the tasks cannot be determined, consequently, the task with the larger cumulative incompletion cost remains to be the first task.

In summary, the sequencing procedure first orders the tasks in the descending order of their cumulative incompletion costs with the precedence constraints observed. Then, each pair of adjacent tasks is examined to find the associated situation described above to determine the relative order of the tasks. If the tasks fit into Situation 3, no action is taken, the task with the larger cumulative incompletion cost remains to be the first task. Note that the relative positions of some pairs of adjacent tasks cannot be changed due to the precedence constraints, although the current positions of the tasks yield a higher expected incompletion cost. Nevertheless, in spite of the fact that the precedence constraints do not allow the formation of every possible sequence, the resequencing process of the tasks in the node may result in a lower expected incompletion cost than that of the sequence formed by the node generation process.

In the next section, the evaluation of the nodes is discussed in detail.

5.3.3 Node Evaluation Scheme

When a node is generated, the tasks in the node are resequenced as discussed in the previous section, and the cost CN associated with the sequence is computed. CN consists of the labor cost of a station and the expected incompletion costs of the tasks in the node. The expression of CN_i corresponding to node i is similar to the return function of the dynamic programming formulation of the problem and is expressed as follows:

$$CN_i = C \times L + \sum_{k \in R_i} \left[\beta_k \left[IC_k + \sum_{j \in A_k} IC_j \right] - \sum_{j=1}^{fs_k} SB_k^j \right]$$

where R_i is the set of tasks in node i. The cost associated with the tasks in the node and in all its parent nodes, designated TCN is computed as follows:

$$TCN_i = TCN_{\zeta_i} + CN_i$$

Let G_i be the set of tasks in node i and in all its parent nodes. That is, $G_i = R_i \cup G_{\zeta_i}$. Then, the relaxed problem corresponding to node i has tasks in the set $W - G_i$. Let N_{ir} be the number of tasks in the set $W - G_i$. In a relaxed problem, precedence constraints among the tasks are relaxed, but the incompletion costs of the tasks are replaced by their cumulative incompletion costs. In other words, the incompletion costs of the tasks are not only proportional to their expected performance times; they are also functions of their positions on the precedence diagram. Thus, the optimal solution of a relaxed problem constitutes a lower bound on the solution of the actual partial problem associated with the node. Let CIN_{iM} denote the expected incompletion cost of the relaxed problem solution of node i obtained by the M-machine scheduling procedure for M stations. Then, the total expected cost of the relaxed problem, CRX_i , can be expressed as follows:

$$CRX_i = \min_{M = 1,\dots, N_{ir}} \{ C L M + CIN_{iM} \}$$

Note that in order to find a good estimate of CRX_i , the M-machine scheduling procedure is applied N_{ir} times to the relaxed problem corresponding to node i for $M = 1, \dots, N_{ir}$. This, in reality, can be quite time consuming. In order to limit the range of this search over the number of stations, an experimentation was conducted to compare the number of stations obtained by the above procedure to that obtained by the dynamic programming procedure with the bounding strategy and the technique of Kottas and Lau. In the problems, $\mu_i \sim U[0; C]$, $\sigma_i = RAN_1 \mu_i$ and $IC_i = RAN_2 \mu_i$ for $i = 1, \dots, N$, and $L = 3.00$ \$/hour. $C \sim U[10; 100]$, $RAN_1 \sim U[0.04; 0.06]$ and $RAN_2 \sim U[L; 2L]$. Table 7 depicts the number of tasks, F-ratios, cycle times, RAN_1 and RAN_2 values of the example problems solved. Table 8 depicts the numbers of stations in the solutions of the example problems using the dynamic programming procedure, technique of Kottas and Lau and the improvement procedure. As it is seen from Table 8, among 16 of the 30 problems, the difference between the number of stations determined by the dynamic programming procedure and the improvement procedure is less than or equal to 1. The difference between the numbers of stations generated by these procedures is, on the average, 2.00. Among 14 of the 30 problems, the difference between the number of stations determined by the technique of Kottas and Lau and the improvement procedure is less than or equal to 1. The difference between the numbers of stations generated by these procedures is, on the average, 2.53. We use these results to guide search over the number of stations. Let $NSTA_{DP}$ denote the number of stations in the initial solution obtained using the dynamic programming procedure. If node i is at level j, then the corresponding relaxed problem is expected to have $NSTA_{DP} - j$ stations. Thus, the above procedure is applied to the relaxed problem for the number of stations in the neighborhood of $NSTA_{DP} - j$. If the difference between the numbers of stations of the solutions of the dynamic programming procedure and the improvement procedure is assumed to be smaller than 5, then the procedure is applied to the relaxed problem associated with node i of level j eleven times for

$M = NSTA_{DP} - j - 5, ..., NSTA_{DP} - j + 5$. This property reduces the computational requirement of the node evaluation process and contributes to the effectiveness of the improvement procedure.

The effectiveness of the improvement procedure comes from several features of the branching and node evaluation schemes. The branching scheme has several features that eliminate the generation of the nodes not leading to the optimal solution. Another important factor for the effectiveness of the improvement procedure is the tight initial solution value obtained from the dynamic programming procedure with α set to 0.5. With a tight initial bound, several nodes are pruned at early stages so that the dimension of the tree grows at a slower rate. The node evaluation scheme utilizes a heuristic procedure to solve the relaxed problems of the nodes of the enumeration tree and the procedure is shown to give good solutions. Thus, the approximate costs of the nodes are quite close to their optimal values; this property identifies the nodes not leading to the optimal solution at the earlier stages of the procedure. The heuristic procedure used to solve the relaxed problems will be presented in the next section. The efficiency of this procedure is further increased by iterating the procedure for a certain number of stations.

5.3.3.1 M-Machine Scheduling Procedure

In this section we will extend the single-machine sequencing rule presented in Section 5.3.2.2 to construct a schedule on M parallel, identical machines. We first review the literature for the problem of scheduling independent tasks on parallel, identical machines. Then, the heuristic procedure of constructing a schedule on M machines is described. Then, some computational experience on the performance of the procedure on some randomly generated problems is reported.

The M-machine scheduling procedure developed in this section is used to solve the relaxed problem at every node of the enumeration tree of the improvement procedure. Thus, the performance of this procedure affects the performance of the improvement procedure significantly. The closer the

Table 7. Parameters of the example problems solved

Example No.	# of tasks	F-ratio	C †	RAN_1	RAN_2
1	11	0.000	58.4	0.052	0.083
2	11	0.000	72.2	0.043	0.076
3	11	0.000	99.9	0.046	0.062
4	11	0.491	23.8	0.057	0.056
5	11	0.491	37.6	0.048	0.099
6	11	0.491	51.4	0.059	0.092
7	11	0.418	65.3	0.051	0.085
8	11	0.418	79.1	0.042	0.079
9	11	0.418	93.0	0.053	0.072
10	11	0.800	57.9	0.052	0.068
11	11	0.800	92.4	0.053	0.058
12	11	0.800	50.8	0.045	0.090
13	15	0.000	99.2	0.057	0.073
14	15	0.000	57.6	0.050	0.055
15	15	0.000	16.0	0.042	0.088
16	15	0.257	64.4	0.054	0.070
17	15	0.257	22.8	0.046	0.053
18	15	0.257	71.2	0.058	0.085
19	15	0.781	29.6	0.050	0.068
20	15	0.781	78.0	0.042	0.050
21	15	0.781	15.6	0.050	0.099
22	16	0.575	84.8	0.046	0.065
23	16	0.575	43.2	0.058	0.098
24	16	0.575	91.6	0.051	0.080
25	17	0.382	50.0	0.043	0.063
26	17	0.382	98.4	0.055	0.095
27	17	0.382	56.8	0.047	0.078
28	18	0.379	15.2	0.059	0.060
29	18	0.379	63.6	0.051	0.093
30	18	0.379	22.0	0.043	0.075

† Cycle times are in minutes

Table 8. Numbers of stations in the solutions of the dynamic programming procedure, the technique of Kottas and Lau and the improvement procedure

Example No.	Number of stations in		
	DP Procedure solution at $\alpha = 0.5$	Kottas and Lau solution	Improvement proc.solution
1	9	9	8
2	7	7	7
3	7	7	3
4	7	8	2
5	5	6	5
6	6	6	6
7	8	9	6
8	7	9	7
9	9	9	7
10	7	8	4
11	7	7	2
12	7	7	5
13	9	9	6
14	10	10	5
15	10	10	10
16	8	8	6
17	10	10	2
18	9	10	9
19	10	10	10
20	8	9	2
21	11	11	11
22	6	7	6
23	8	10	8
24	9	9	9
25	9	10	6
26	10	11	10
27	9	10	9
28	11	11	2
29	7	8	7
30	11	12	11

solutions of this procedure are to the optimum, the better will be the quality of the final solution of the problem.

Related Literature On The M-Machine Scheduling Problem: The problem of scheduling independent tasks on parallel, identical machines was first considered by McNaughton [66]. He developed rules for minimizing the total completion time where tasks may be split among machines. Later, several researchers examined the problem for different performance measures. Root [83] considered the minimization of the penalty cost of the total tardiness. The penalty cost was assumed to be a linear function of tardiness and a common due date was applied to all tasks. Gupta and Maykut [36] and Rothkoph [84] presented dynamic programming formulations of the problem. Elmaghraby and Park [30] considered a branch-and-bound algorithm for finding the optimal schedule for the objective of minimizing the penalty cost of total tardiness. The penalty cost was any nondecreasing function of the tardiness. Dogramaci and Surkis [27] considered a heuristic procedure to minimize total tardiness.

In view of the complexity of the problem, work has focused on finding heuristic procedures. The measure of performance of a heuristic procedure is the ratio of the solution value obtained using the heuristic procedure to that of the optimal solution value. Sarin and Elmaghraby [86] proposed a heuristic procedure for the criterion of minimization of the total weighted completion times. They derived bounds on the worst-case performance of the procedure. Loulou [62] obtained upper bounds on the difference of the values of the heuristic solution and the optimal solution, whereas Graham [35] and Garey and Graham [33] obtained bounds on the ratio of the values of the heuristic solution and the optimal solution for the objective of minimizing the makespan. Coffman and Gilbert [18] obtained bounds on the ratio of the values of the heuristic solution and the optimal solution where the task performance times were chosen from a uniform distribution or an exponential distribution for the objective of minimizing the makespan. Bruno and Downey [10] determined the expression such that the probability of the ratio of the values of the heuristic solution and the optimal solution being less than the expression is greater than a prespecified value. The objective was to minimize the makespan, and task performance times were chosen from a uniform distribution. Eastman, Even and Isaacs [29] derived lower and upper bounds on the cost of an optimal schedule for the problem with the objective of minimizing total weighted flowtimes.

Our problem is different in the sense that a fixed cost is incurred for each task while the incompletion probability depends on the position of the task in the sequence and its expected processing time.

Development Of The Heuristic Procedure: Consider M parallel, identical machines with N tasks waiting. Assume that the machines are free at the moment, and we wish to allocate tasks among the machines and then sequence them on each machine. The performance measure to be optimized is the expected incompletion cost. The performance measure can be expressed as:

$$\text{Min} \sum_{i=1}^{N} IC_i \times \Pr\{ f_i > C \}$$

where f_i is the flowtime of task i. Each machine can process only one task at a time, and having once started a task, finishes it before starting another. Splitting of tasks among the machines is not permitted. All the assumptions of the procedure of sequencing tasks on a single machine described in Section 4.3.2.2 also apply to this procedure.

The scheduling procedure first forms a single-machine sequence with the due date set to M × C using the single-machine sequencing procedure presented in Section 5.3.2.2. Allocation of the tasks to the M machines is achieved in the order of the appearance of the tasks in the single-machine sequence. Tasks are assigned to the machine that has the least sum of the expected processing times of the tasks already assigned to it. After scheduling the tasks among the M machines, they are resequenced within each machine using the single-machine sequencing procedure. The steps of the procedure can be outlined as follows:

Step 1. Constructing the single-machine sequence. Set the due date equal to $M \times C$. Order the tasks in the descending order of their cumulative incompletion costs. Examine each pair of adjacent tasks to find the associated situation as described in Section 5.3.2.2 and determine the relative order of the tasks. Continue until all pairs of adjacent tasks are examined.

Step 2. Allocation of the tasks to M machines. Allocate tasks to M machines sequentially in their order of appearance in the single machine sequence by assigning the next task to the machine that has the least sum of the expected processing times of the tasks already assigned to it. Continue until all the tasks are assigned.

Step 3. Resequencing of the tasks on the machines. The tasks within each machine are resequenced according to the single machine sequencing procedure.

Step 3 is applied to the schedule generated in Step 2, since the individual machine sequences obtained in Step 2 may not satisfy the single-machine sequencing rule. Thus, the expected incompletion costs of the individual machine sequences can be decreased.

Performance Of The Heuristic: The M-machine scheduling procedure presented above does not guarantee the generation of the optimal solution. Let the solution of the overall problem obtained using this procedure at every node be an ε-optimal solution, i.e., it is within $(1 + \varepsilon)$ of the optimal solution. Also, if CRX_i is the solution of the relaxed problem of node i obtained using this procedure, then there exists a $\rho_{i,}\ \rho_i > 0$, such that $\frac{CRX_i}{1 + \rho_i}$ is a valid lower bound for node i. Consequently, if $\frac{CRX_i}{1 + \rho_i}$ is used, instead of APP_i, at node i, then the enumeration procedure will guarantee the generation of the optimal solution. Now, ε and ρ_i are not known apriori. However, if ρ_i is taken to represent the ρ_i-optimality of the approximation procedure used to solve the relaxed problem at node i, then $\varepsilon = \max_{i=1,\dots,n} \rho_i$, since in the worst case, one of the nodes leading to the optimal solution can have the maximum ρ value. This, in fact, suggests the following way to estimate the value of ε experimentally. A problem is solved using the same ρ value for all the nodes of the enumeration tree of the improvement procedure. If the ρ value assumed is not sufficiently large, then increasing its value should improve the solution. On the other hand, since the procedure to solve the relaxed problems is a heuristic, and approximate costs are used instead of upper bounds for pruning nodes, increasing the ρ value does not necessarily improve the solution, but on the average, an improvement in the solutions is expected. If the improvement in the solutions stabilizes beyond a ρ value, it implies that the value of ρ is either larger than ε of the scheduling procedure or it is very close to it. Therefore, in order to investigate the value of ε, several randomly generated problems were solved with the improvement procedure assuming different ρ values. The parameters of the problems are depicted in Table 7. The initial solutions of the problems were obtained using the dynamic programming procedure with α set to 0.5. Six different ρ values were assumed; namely, 0.0, 0.1, 0.2, 0.3, 0.4 and 0.5, and CRX_i was replaced by $\frac{CRX_i}{1 + \rho}$ for $i = 1,...,n$.

It is highly unlikely that the ε value corresponding to a problem is larger than 0.5. We solved 30 problems using the dynamic programming procedure with α set to 0.5, the technique of Kottas and Lau and the improvement procedure. In all these problems, improvement procedure consistently generated better solutions than the other procedures. The technique of Kottas and Lau is reported in the literature to generate good solutions and the dynamic programming procedure with the bounding strategy is also found to generate good solutions as discussed in Section 4.5. The solutions of the 30 problems obtained using the above procedures are depicted in Table 14. Thus, this evidence indicates that conducting the experimentation for ρ values smaller that 0.5 should cover all the cases. In other words, it is highly unlikely that any improvement in the solution of a problem can be achieved for $\rho > 0.5$. Table 9 depicts the improvement procedure solutions of the problems assuming different ρ values. Table 10 depicts the percentage improvement of the solutions over the case with $\rho = 0.0$. The average percentage improvements of the solutions with ρ values of 0.1, 0.2, 0.3, 0.4 and 0.5 over the case $\rho = 0.0$ are 0.68, 0.92, 1.00, 1.05 and 1.21, respectively. As it is seen, the average percentage improvements are quite small. Assuming that the increase in the improvement stabilizes, on the average, for $\rho > 0.1$, the M-machine scheduling procedure generates, on the average, solutions within 110 percent of the optimal solution. In fact, considering the fact that the average improvement achieved by increasing ρ from 0.0 to 0.1 is 0.68

percent, we can conclude that the M-machine scheduling procedure generates almost optimal solutions.

Although the percentage improvements achieved by increasing ρ are negligible, the increases in the storage and computational requirements of the procedure are significant. The CPU time required and the total number of nodes generated in the tree of the procedure for different ρ values are depicted in Tables 11 and 12, respectively. As it is seen from the tables, the increases in the storage and computational requirements are incomparably larger than the improvements in the solution values. Note that the relaxed problems corresponding to the nodes of the enumeration tree are solved by applying the M-machine scheduling procedure for all possible number of stations.

The experimentation performed indicates that the M-machine scheduling procedure results in solutions quite close to the optimal ones. Hence, it constitutes an effective procedure to solve the relaxed problems corresponding to the nodes of the enumeration tree formed by the improvement procedure. In addition, the storage and computational requirements of the procedure are also quite small.

In the following section, the computer implementation of the improvement procedure will be discussed. A conceptual flowchart of the program will also be presented.

5.3.4 Computer Implementation Of The Improvement Procedure

The listing of the computer program written for the improvement procedure is given in Appendix B. The program is written in FORTRAN and can handle problems of up to 100 tasks. The maximum number of nodes that can be generated in the tree is set at 10,000 and the maximum CPU time that a problem can take is limited to 120 seconds. The dynamic programming procedure solution is fed into the program in addition to the required data by the dynamic programming procedure program. The number of stations in the initial solution and the range of the station

Table 9. Improvement procedure solutions of the example problems for different ρ values

Example	Solution of the improvement procedure for ‡					
	ρ = 0.0	ρ = 0.1	ρ = 0.2	ρ = 0.3	ρ = 0.4	ρ = 0.5
1	32.539	32.539	32.539	32.539	32.539	32.539
2	25.372	25.372	25.372	25.372	25.372	25.372
3	30.496	30.496	30.496	30.471	30.471	30.471
4	7.963	7.963	7.963	7.963	7.963	7.963
5	9.930	9.930	9.930	9.930	9.930	9.930
6	15.487	15.487	15.440	15.440	15.440	15.440
7	36.542	36.542	36.542	36.542	36.542	36.542
8	44.041	43.028	43.028	41.538	41.538	41.448
9	41.556	41.556	41.556	41.556	41.556	41.547
10	21.513	20.499	20.413	20.408	20.405	20.405
11	26.049	26.049	26.049	26.049	26.049	26.049
12	19.741	19.586	19.586	19.586	19.586	19.586*
13	55.109	55.109	55.109	55.109	55.109	55.103
14	25.433	25.433	25.433	25.433	25.433	25.433
15	10.063	10.002	10.002	10.002	10.002	10.002
16	25.713	25.482	25.482	25.482	25.482	25.482
17	9.388	9.388	9.388	9.388	9.388	9.388
18	32.107	32.107	32.107	31.449	31.449	31.449
19	14.938	14.938†				
20	30.076	28.762	28.762	28.762	28.762	28.762†
21	8.644	8.058	8.058†			
22	29.038	29.038	29.038	28.923	28.923	28.923
23	19.348	19.348	19.348†			
24	52.219†					
25	27.588	27.588	26.567	26.567	26.567	26.285
26	71.919	71.919	71.919	71.919†		
27	25.928	25.928	25.928	25.928	25.928†	
28	7.159	7.159	7.159†			
29	27.246	27.246	27.246	27.246	27.246†	
30	13.688	13.688†				

† Procedure exceeded the CPU time limit of 120 seconds
* Procedure exceeded the storage limit of 10,000 nodes
‡ Solution values are in $/unit

Table 10. Percentage improvement of the solutions of the example problems over the case when $\rho = 0.0$

Example	Percentage improvement achieved using				
	$\rho = 0.1$	$\rho = 0.2$	$\rho = 0.3$	$\rho = 0.4$	$\rho = 0.5$
1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.1	0.1	0.1
4	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0
6	0.0	0.3	0.3	0.3	0.3
7	0.0	0.0	0.0	0.0	0.0
8	2.3	2.3	5.7	5.7	5.9
9	0.0	0.0	0.0	0.0	0.0
10	4.7	5.1	5.1	5.2	5.2
11	0.0	0.0	0.0	0.0	0.0
12	0.8	0.8	0.8	0.8	0.8*
13	0.0	0.0	0.0	0.0	0.0
14	0.0	0.0	0.0	0.0	0.0
15	0.6	0.6	0.6	0.6	0.6
16	0.9	0.9	0.9	0.9	0.9
17	0.0	0.0	0.0	0.0	0.0
18	0.0	0.0	2.1	2.1	2.1
19	0.0†				
20	4.4	4.4	4.4	4.4	4.4†
21	6.8	6.8†			
22	0.0	0.0	0.4	0.4	0.4
23	0.0	0.0†			
24	†				
25	0.0	3.7	3.7	3.7	4.7
26	0.0	0.0	0.0†		
27	0.0	0.0	0.0	0.0†	
28	0.0	0.0†			
29	0.0	0.0	0.0	0.0†	
30	0.0†				

† Procedure exceeded the CPU time limit of 120 seconds

* Procedure exceeded the storage limit of 10,000 nodes

Table 11. CPU time used by the improvement procedure for the solutions of the example problems for different ρ values

Example	CPU time spent in the procedure for ‡					
	$\rho = 0.0$	$\rho = 0.1$	$\rho = 0.2$	$\rho = 0.3$	$\rho = 0.4$	$\rho = 0.5$
1	0.15	0.15	0.17	0.19	0.19	0.20
2	0.05	0.14	0.16	0.16	0.17	0.17
3	0.11	0.11	0.11	0.20	0.20	0.20
4	0.21	0.28	0.28	0.28	0.36	0.36
5	0.04	0.04	0.04	0.04	0.04	2.32
6	0.04	0.04	1.23	2.73	4.70	5.68
7	2.15	2.91	3.58	4.26	5.39	6.82
8	0.04	1.03	1.59	2.01	2.56	3.42
9	0.95	2.61	4.42	5.75	6.52	7.16
10	7.01	13.03	31.24	54.74	83.95	111.07
11	0.98	2.34	3.60	4.26	5.69	6.69
12	0.78	8.27	25.76	63.41	95.45	104.63*
13	0.33	0.34	0.34	0.37	0.37	0.49
14	0.28	0.30	1.49	1.67	1.67	1.67
15	0.25	0.44	0.48	0.54	0.60	0.63
16	0.71	0.77	1.13	1.49	2.92	2.96
17	9.78	9.78	9.78	9.78	9.78	9.78
18	0.07	0.06	0.69	2.07	3.61	6.14
19	2.54	120.01†				
20	2.72	10.51	23.69	58.14	104.83	120.03†
21	0.29	65.05	120.01†			
22	1.26	6.09	26.34	45.44	76.60	120.01†
23	0.07	84.09	120.01†			
24	120.01†					
25	1.48	2.93	1.74	6.58	13.80	21.28
26	0.13	0.13	60.92	120.01†		
27	0.10	0.10	0.14	51.22	120.01†	
28	13.62	53.74	120.09†			
29	0.09	0.09	19.51	80.70	120.01†	
30	0.12	120.01†				

† Procedure exceeded the CPU time limit of 120 seconds
* Procedure exceeded the storage limit of 10,000 nodes
‡ CPU times are in seconds

Table 12. Number of nodes generated by the improvement procedure for the solutions of the example problems for different ρ values

Example	Number of nodes generated by the improvement procedure using					
	ρ = 0.0	ρ = 0.1	ρ = 0.2	ρ = 0.3	ρ = 0.4	ρ = 0.5
1	12	12	16	18	18	22
2	1	7	9	9	10	10
3	4	4	4	8	8	8
4	1	2	2	2	3	3
5	1	1	1	1	1	182
6	1	1	104	203	363	553
7	151	219	346	445	640	877
8	1	42	85	107	175	286
9	74	228	421	624	739	913
10	193	367	1101	2188	4342	6187
11	4	23	62	97	146	190
12	18	699	2050	4669	7936	9996*
13	8	8	8	10	10	15
14	8	8	14	18	18	18
15	6	24	27	35	43	45
16	14	19	36	50	131	134
17	3	4	4	4	4	4
18	1	1	19	87	164	324
19	27	2533†				
20	4	11	46	128	339	469†
21	1	1675	2300†			
22	4	50	347	711	1327	3502†
23	1	1518	4062†			
24	5582†					
25	17	44	23	64	187	309
26	2	2	2676	8198†		
27	1	1	2	1997	6031†	
28	4	33	22†			
29	1	1	399	1439	2536†	
30	1	2214†				

† Procedure exceeded the CPU time limit of 120 seconds

* Procedure exceeded the storage limit of 10,000 nodes

numbers at which the relaxed problems are solved should also be provided. A conceptual flowchart of the program is depicted in Figure 15.

In the computer program, subroutine NODGEN checks if there are any unexplored nodes left in the tree and determines the node with the least approximate cost to branch into the nodes of the next level. Subroutine GNRTOR generates the descendent nodes of a node using the node generation process described in Section 5.2.2. Function MATRIX is used to augment the unmarked immediate followers of a state to the current state. The tasks of the new nodes are resequenced with subroutine DIZI. Subroutine ALTSNR applies the M-machine scheduling procedure to the relaxed problems of the nodes. Tasks are scheduled among the M machines with subroutine SCHDL. Subroutine PROB calculates the incompletion probabilities of the tasks. Finally, subroutine EVAL determines if the nodes generated are feasible.

5.4 Concluding Remarks On The Approximation Procedure

Various features of the approximation procedure are discussed and the procedures used by the improvement procedure are developed in the sections above. Before applying the approximation procedure, the problem should be divided into subproblems with N_{sp} or less number of tasks as described in Section 5.1.1. The value of N_{sp} should be known before dividing the problem into subproblems. Based on the experimentation conducted to investigate the performance of the M-machine scheduling procedure, we conclude that N_{sp} should be 20. First of all, the storage and computational requirements of the dynamic programming procedure with the bounding strategy used to obtain the initial solutions of the improvement procedure remain at relatively small values

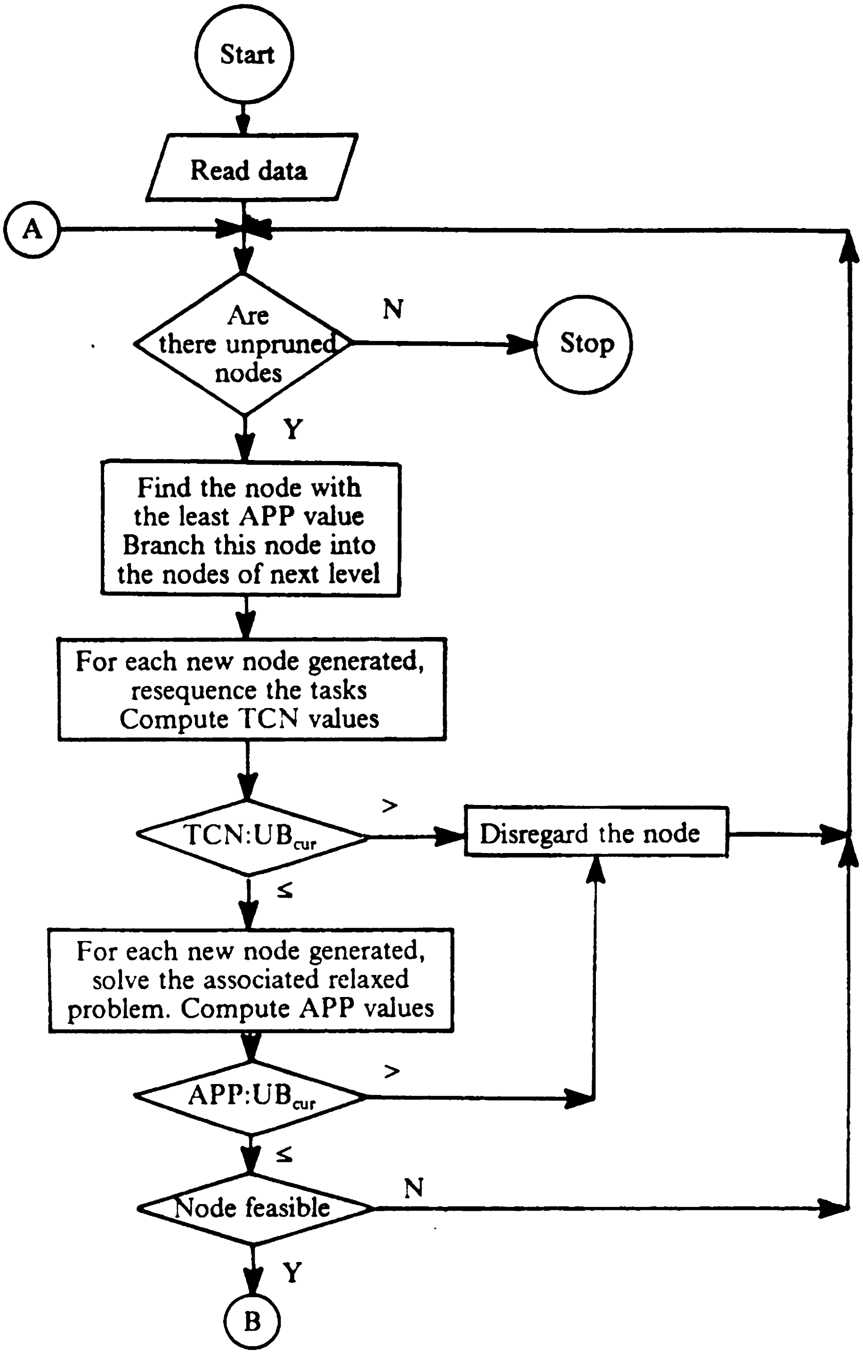
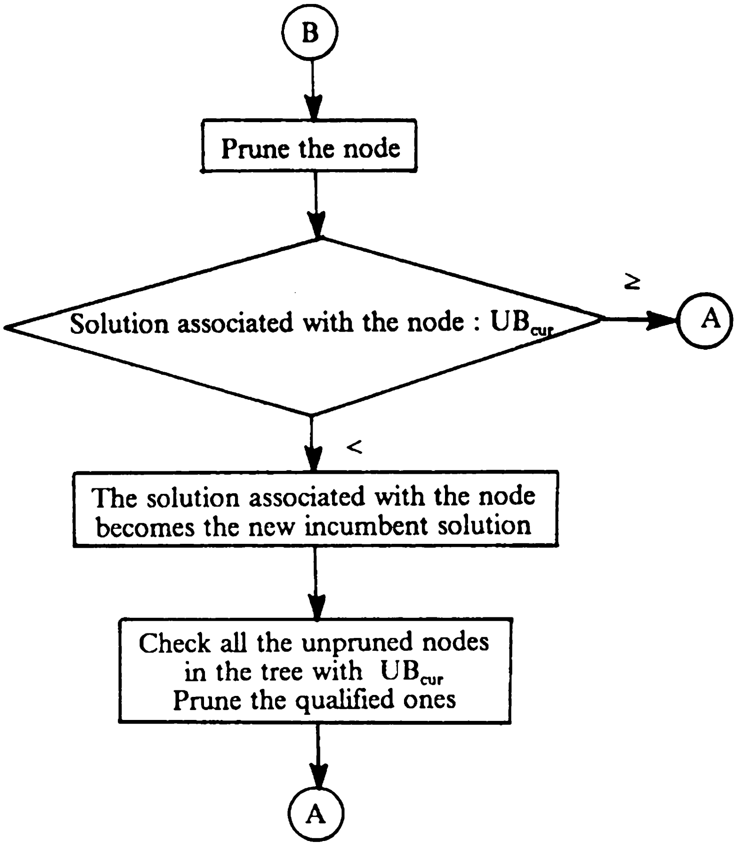

Start
Read data
A
Are there unpruned nodes
N
Stop
Y
Find the node with the least APP value Branch this node into the nodes of next level
For each new node generated, resequence the tasks Compute TCN values
TCN:UB$_{cur}$
>
Disregard the node
≤
For each new node generated, solve the associated relaxed problem. Compute APP values
APP:UB$_{cur}$
>
≤
Node feasible
N
Y
B

Figure 15. Conceptual flowchart of the improvement procedure


B
Prune the node
Solution associated with the node : UB$_{cur}$
≥
A
<
The solution associated with the node becomes the new incumbent solution
Check all the unpruned nodes in the tree with UB$_{cur}$ Prune the qualified ones
A

(Figure 15 continued)

for problems with 20 or less number of tasks. Table 13 depicts the storage and computational requirements of the dynamic programming procedure for the example problems depicted in Table 7. Secondly, the improvement procedure also requires relatively small amounts of storage and computation for problems with 20 or less number of tasks as depicted in Tables 11 and 12.

The improvement procedure generates solutions within $(1 + \varepsilon)$ of the optimal solutions of the subproblems. Thus, the final solution of the problem after appending the subproblem solutions at Step 4 of the approximation procedure should also be very close to the optimal one. To investigate this property of the approximation procedure and some other aspects of the approximation and improvement procedures, several randomly generated problems are solved. The details of the experimentation and a discussion of the results will be presented in the next section.

5.5 Computational Experience

In this section the results of the experimentation performed to examine several aspects of the dynamic programming procedure with α set to 0.5, the improvement procedure and the approximation procedure will be reported. A comparative discussion of the procedures based on the outcomes will also be presented.

Two sets of problems are solved with the approximation procedure. The first set of problems are those which do not require decomposition and the second set of problems are those that require decomposition of the given problems into subproblems (based on the analysis reported in Section 5.4). The problems in the first set have 18 or less tasks, while those in the second set have more than 18 tasks. One of the purposes of studying the second set of problems is to see the effect of decomposition on the quality of the solutions. The problems investigated have the following parameters: $\mu_i \sim U[0, C]$ with $\sigma_i = RAN_1(\mu_i)$ and $IC_i = RAN_2(\mu_i)$ and $L = 3.00$ \$/hour.

Table 13. Storage and computational requirements of the dynamic programming procedure with the bounding strategy for the example problems

Example	# of cumulative decision vars.	# of cumulative state vars.	CPU time †
1	30	27	0.44
2	50	33	0.48
3	61	37	0.49
4	271	166	1.63
5	314	244	2.41
6	284	237	2.13
7	153	138	0.94
8	123	111	0.74
9	94	133	0.73
10	397	688	9.84
11	447	734	11.05
12	391	747	11.53
13	91	57	0.70
14	94	61	0.71
15	74	51	0.67
16	359	192	2.01
17	167	166	1.02
18	252	170	1.40
19	598	1751	47.07
20	1504	2076	112.20
21	524	1544	38.30
22	2788	1269	103.69
23	1246	991	26.88
24	1513	1151	36.23
25	770	553	9.73
26	862	550	10.74
27	810	584	10.99
28	901	741	11.63
29	1175	737	13.15
30	568	568	5.66

† CPU times are in seconds

RAN_1 and RAN_2 are the multipliers used to obtain the variance and the incompletion costs of the tasks, respectively, and reported in Table 7. Tables 7 and 14 depict the parameters and the solutions of the problems in the first set, respectively. Tables 21 and 22 depict the corresponding values of the problems in the second set, respectively.

In Table 14, Kottas and Lau solution column and the dynamic programming solution at α_1 column give the solution values of the problems solved using the technique of Kottas and Lau and the dynamic programming procedure with α set to α_1, respectively. α_1 is set to 0.5, since $\alpha \leq 0.5$, because $\mu_i \sim U\,[0; C]$ for all i. The improvement procedure solution column gives the solution values obtained using the improvement procedure. The dynamic programming procedure solutions provide the initial solutions to the improvement procedure. The last two columns of the Table depict the percentage difference between the improvement procedure solution values with those of the Kottas and Lau technique and the dynamic programming procedure. A comparison of the dynamic programming procedure solution values with the bounding strategy and those of the Kottas and Lau technique reveals that the dynamic programming procedure results in solutions as good as the technique of Kottas and Lau. The performance of the dynamic programming procedure depends on the value of α, the larger the value of α, the better the performance of the dynamic programming procedure. Only in 1 of the 30 problems (namely, problem 19) reported in Table 14, the dynamic programming procedure performed worse than the Kottas and Lau technique. On the average, the dynamic programming procedure results in solutions with values 5.0 percent lower than those of the Kottas and Lau technique; 90% confidence interval limits on this value are 3.3 percent and 6.7 percent, respectively. A comparison of the improvement procedure solution values and those of the Kottas and Lau technique reveals that the improvement procedure results in better solutions for all the problems. The average improvement in the solutions is 9.6 percent, and the 90% confidence interval limits on the improvement in the solutions of the problems are 7.5 percent and 11.7 percent, respectively. A comparison of the improvement procedure solution values and those of the dynamic programming procedure with the bounding strategy indicates how much the initial solutions are improved by the improvement procedure, since the dynamic programming procedure provides the initial solutions to the improvement procedure. The average improvement achieved on the initial solutions is 4.7 percent, and 90% confidence interval limits on this improvement are 2.7 percent and 6.7 percent, respectively.

Next, we describe how the technique of Kottas and Lau and the dynamic programming procedure with the bounding strategy work. The technique of Kottas and Lau is a single-pass technique; in other words, once a decision is made to assign a task to a station, the task is never considered again. While further improvement could be made by reconsidering the task for assigning to a different station. In addition, the marginal desirability of a task is determined only by examining the expected performance time, incompletion cost and the position of the task in the station. Note that a task can be started only if its predecessors are completed. Therefore, the probability that a task under consideration is started should also be a factor in determining the marginal desirability of the task. These factors contribute to the deviation of the solutions from the optimal ones. On the other hand, the deviation of the dynamic programming procedure with the bounding strategy solutions from the optimal ones comes from the magnitude of α at which the procedure is applied. As noted above, increasing the value of α would improve the solutions obtained by the dynamic programming procedure. In summary, due to the reasons discussed above, the technique of Kottas and Lau and the dynamic programming procedure with the bounding strategy solutions deviate from the optimal ones, and the deviation for both of these procedures are quite close to each other.

To determine the quality of the solutions generated by the improvement procedure relative to the optimal ones, we applied the approximation procedure to the problems solved in Section 4.5 for which the optimal solutions were obtained. Table 15 depicts the results in a similar format with Table 5; OPT($\alpha = 0.5$) values of Table 5 are replaced with the approximation procedure solutions. Table 15 reveals that the approximation procedure results in solutions quite close to the optimal ones; the ratio of the approximation procedure solution and the optimal solution ranges between 1.00 and 1.16, the average ratio for the 15 problems is approximately 1.06. On the other hand, the ratios in Table 5 range between 1.00 and 1.35, the average being approximately 1.12.

Table 14. Solutions of the example problems with the technique of Kottas and Lau, dynamic programming procedure and improvement procedure

				percentage difference between imp.procedure solution and	
Example No.	Kottas and Lau solution †	DP procedure solution at α_1 †	Imp.procedure solution †	Kottas and Lau solution	DP procedure solution
1	34.420	34.420	32.539	5.5	5.5
2	25.372	25.372	25.372	0.0	0.0
3	35.842	35.842	30.496	14.9	14.9
4	9.880	8.833	7.963	19.4	9.9
5	11.810	9.930	9.930	15.9	0.0
6	15.576	15.487	15.487	0.6	0.0
7	41.202	37.938	36.542	11.3	3.7
8	50.425	44.041	44.041	12.7	0.0
9	42.873	42.873	41.556	3.1	3.1
10	25.346	22.446	21.513	15.1	4.2
11	33.264	32.595	26.049	21.7	20.1
12	19.946	19.937	19.741	1.0	1.0
13	62.104	61.753	55.109	11.3	10.8
14	28.828	28.806	25.433	11.8	10.7
15	10.064	10.063	10.063	0.0	0.0
16	25.799	25.766	25.713	0.3	0.2
17	11.434	11.430	9.388	17.9	17.9
18	36.375	32.107	32.107	11.7	0.0
19	15.462	15.880	14.938	3.4	5.9
20	37.575	33.811	30.076	20.0	11.1
21	8.831	8.644	8.644	2.1	0.0
22	33.007	29.038	29.038	12.0	0.0
23	22.537	19.348	19.348	14.2	0.0
24	52.219	52.219	52.219	0.0	0.0
25	31.654	29.114	27.588	12.9	5.2
26	75.012	71.919	71.919	4.1	0.0
27	28.734	25.928	25.928	9.8	0.0
28	8.580	8.563	7.159	16.6	16.4
29	30.085	27.246	27.246	9.4	0.0
30	14.851	13.688	13.688	7.8	0.0

† Solution values are in $/unit

Table 15. Comparison of the approximation procedure solutions with the optimal ones

F-ratio	Number of tasks	App.procedure solution	OPT*	$\frac{\text{App.procedure sol.}}{\text{OPT}^*}$
0.000	10	30.557	27.318	1.119
0.000	10	21.621	19.207	1.126
0.000	10	25.950	25.950	1.000
0.000	10	27.385	26.807	1.022
0.000	10	7.263	6.786	1.070
0.000	15	9.928	8.727	1.138
0.000	15	15.795	13.586	1.163
0.000	15	17.207	16.695	1.031
0.000	15	35.163	34.947	1.006
0.000	15	42.263	40.353	1.047
0.418	11	9.948	9.948	1.000
0.418	11	21.153	20.251	1.045
0.418	11	26.859	25.500	1.053
0.418	11	22.007	22.007	1.000
0.418	11	19.102	19.102	1.000

Considering the second set of problems, Table 16 depicts the number of tasks, F-ratios, cycle times, RAN_1 and RAN_2 values. The number of tasks of these problems range from 30 to 60; hence they are decomposed into subproblems of 20 or less number of tasks as described in Section 5.1. The precedence diagrams of the last 12 problems in the Table are generated randomly as follows: There is a precedence relation between tasks i and j if a random number generated is greater than a test value of b. Thus, precedence relations among the tasks are determined by generating $\frac{N(N+1)}{2}$ random numbers. Labels of the tasks are then determined and they are numbered as described in Section 5.1. The test value, b is changed between 0.4 and 0.6 to generate various forms of precedence diagrams.

Table 17 depicts the solution values obtained using the approximation procedure for the 18 problems whose parameters are given in Table 16. The approximation procedure solution value column depicts the solution after the improvement procedure is applied to the subproblems. Note that the initial solutions of the improvement procedure for the subproblems are obtained with the dynamic programming procedure with α set to 0.5. The CPU time limit on the improvement procedure of each subproblem is set to 120 seconds. The total CPU time taken by the procedure is depicted in the next column. It consists of the time required by the dynamic programming procedure to provide the initial solutions and the improvement procedure of the subproblems. Note that the solutions are obtained using the IBM 3090, and the computer programs are compiled at compiler optimization level of 3. The last column gives the percentage improvement made over the solution of the technique of Kottas and Lau. On the average, the approximation procedure generated designs with total system costs 6.2 percent lower than those of the designs generated by the technique of Kottas and Lau; 90% confidence interval limits on this improvement are 4.2 percent and 8.2 percent, respectively. The approximation procedure performed better than the technique of Kottas and Lau for all the problems; the maximum improvement achived was 15.7 percent. Based on these results, we conclude that the approximation caused by decomposing the problem into subproblems does not seem to affect the quality of the final solution significantly.

Table 16. **Parameters of the example problems solved**

Example No.	# of tasks	F-ratio	C †	RAN$_1$	RAN$_2$
31	30	0.372	35.6	0.052	0.055
32	30	0.372	84.0	0.044	0.088
33	30	0.372	42.4	0.056	0.070
34	40	0.173	90.8	0.048	0.053
35	40	0.173	49.2	0.040	0.085
36	40	0.173	97.6	0.052	0.068
37	30	0.051	56.0	0.044	0.050
38	30	0.051	14.4	0.056	0.083
39	30	0.051	62.8	0.049	0.065
40	30	0.147	76.4	0.057	0.095
41	30	0.147	34.8	0.049	0.078
42	30	0.147	83.2	0.041	0.060
43	50	0.074	41.7	0.053	0.093
44	50	0.074	90.0	0.045	0.075
45	50	0.074	48.5	0.057	0.058
46	60	0.057	96.9	0.050	0.090
47	60	0.057	55.3	0.042	0.073
48	60	0.057	13.7	0.054	0.055

† Cycle times are in minutes

Table 17. Solutions obtained by the approximation procedure and the technique of Kottas and Lau for the example problems that are decomposed into subproblems

		Approximation procedure		
Example No.	Kottas and Lau solution †	Solution value †	CPU time taken ‡	Percentage improvement over Kottas and Lau solution
31	35.655	33.126	5.73	7.1
32	82.222	78.872	9.96	4.1
33	61.452	61.338	124.83	0.2
34	166.947	140.716	147.02	15.7
35	75.875	70.952	48.33	6.5
36	150.763	150.639	254.87	0.1
37	78.858	70.371	62.55	10.8
38	16.883	16.251	28.55	3.7
39	46.728	45.155	11.12	3.4
40	84.470	78.420	19.47	7.2
41	46.062	42.578	124.79	7.6
42	90.035	78.162	131.91	13.2
43	97.969	95.127	50.60	2.9
44	226.199	223.261	13.26	1.3
45	106.564	90.746	156.22	14.8
46	301.845	292.288	131.23	3.2
47	142.366	141.807	251.05	0.4
48	29.130	26.427	380.79	9.3

† Solution values are in $/unit

‡ CPU times are in seconds

5.5.1 Study of the Impact of the Magnitude of the Incompletion Costs Relative to the Labor Rate on Quality of the Solutions

An important factor in generating the type of solution is the magnitude of the incompletion costs of the tasks relative to the labor rate. If the magnitude of the incompletion cost is small relative to the labor rate, then more tasks will be assigned to a typical station in order to minimize the total system cost, because it will cost less to complete tasks off the line. That is, relatively, a larger number of tasks will be incomplete. On the other hand, if the magnitude of the incompletion cost is large relative to the labor rate, then a greater number of tasks will be completed on the line, thereby utilizing more stations. An experimentation was conducted to study the impact of the relative magnitude of the incompletion costs on the quality of the solutions generated by the dynamic programming procedure with the bounding strategy, Kottas and Lau technique and the improvement procedure. The results of this experimentation are presented in Tables 18, 19, 20 and 21.

In the experimentation, the relative magnitude of the incompletion cost is determined by the multiplier RAN_2, where $IC_i = RAN_2 \times \mu_i$ for all i. RAN_2 is varied from L to 4L with increments of L, where L is the labor rate. Tables 18, 19, 20 and 21 contain results for $RAN_2 =$ L, 2L, 3L and 4L, respectively. 90% confidence interval limits on the percentage differences between the solutions of the problems reported in these Tables are summarized in Table 22. Also, the results for the case of $RAN_2 \sim U[L, 2L]$ are reported in Table 22 which are based on the percentage differences reported in Table 14. Note that the average percentage difference between the solution values of the improvement procedure and those of the Kottas and Lau technique and the dynamic programming procedure are different for different values of RAN_2. In particular, as RAN_2 increases, the percentage differences decrease. The percentage difference values are higher at the lower values of RAN_2, because both Kottas and Lau technique and the dynamic programming procedure utilize larger number of stations than that utilized by the improvement procedure. Based on the observations and the discussion in the previous paragraph, we can conclude that Kottas and Lau technique and the dynamic programming procedure utilize larger number of stations in the solutions at lower values of RAN_2 than needed to minimize the total system cost. However, as RAN_2 increases, the number of stations utilized in the solutions generated by these procedures tend to be close to those utilized by the improvement procedure, even though the improvement procedure continues to give better solutions (See Table 22). Hence, this shows that the Kottas and Lau technique and the dynamic programming procedure tend to be insensitive to the relative magnitude of the incompletion costs, whereas the improvement procedure gives solutions that are very close to the optimum ones for all relative magnitudes of incompletion costs. Furthermore, it can be inferred that both Kottas and Lau technique and the dynamic programming procedure perform better for large relative magnitudes of incompletion costs (large RAN_2 values) than at the lower values.

5.5.2 Conclusions On The Computational Experience

In this section the experimentation conducted to examine various features of the approximation and improvement procedures is presented. The conclusions drawn can be summarized as follows:

1. The approximation procedure generated solutions at least as good as the technique of Kottas and Lau, which is a well-known technique having the same objective function. In most of the problems, the approximation procedure resulted in far better solutions.

2. The approximation procedure required more computing time than the technique of Kottas and Lau.

3. The dynamic programming procedure with the bounding strategy could be used to solve problems of moderate sizes. The dynamic programming procedure results at least as good solutions as the technique of Kottas and Lau.

Table 18. Solutions of the example problems with the technique of Kottas and Lau, dynamic programming procedure and improvement procedure for $RAN_2 = L$

Example No.	Kottas and Lau solution †	DP procedure solution at α_1 †	Imp.procedure solution †	percentage difference between imp.procedure solution and	
				Kottas and Lau solution	DP procedure solution
1	28.923	31.045	20.427	29.4	34.2
2	25.342	25.342	18.978	25.1	25.1
3	35.671	35.671	25.995	27.1	27.1
4	9.843	8.781	7.408	24.7	15.6
5	11.549	9.669	9.669	16.3	0.0
6	15.511	15.462	12.712	18.1	17.8
7	36.311	33.047	24.852	31.6	24.8
8	45.043	38.104	28.307	37.2	25.7
9	42.558	42.558	33.950	20.2	20.2
10	25.346	22.446	21.513	15.1	4.2
11	31.203	32.562	26.666	14.5	18.1
12	18.984	18.979	14.827	21.9	21.9
13	56.672	56.430	44.773	21.0	20.7
14	28.826	28.806	24.414	15.3	15.3
15	9.181	9.181	7.720	15.9	15.9
16	26.025	25.766	21.771	16.4	15.5
17	11.433	11.429	9.036	21.0	20.9
18	33.890	32.082	26.684	21.3	16.8
19	15.291	15.601	13.129	14.1	15.9
20	37.569	33.962	30.017	20.1	11.6
21	8.712	8.618	6.941	20.3	19.5
22	32.236	28.203	25.936	19.5	8.0
23	22.765	18.343	15.996	29.7	12.8
24	48.085	48.085	35.814	25.5	25.5
25	28.031	27.779	20.020	28.6	27.9
26	65.102	61.142	45.278	30.5	26.0
27	28.619	25.801	22.496	21.4	12.8
28	8.545	8.531	6.548	23.4	23.2
29	27.951	24.954	22.095	21.0	11.5
30	14.304	13.161	10.498	26.6	20.2

† Solution values are in $/unit

Table 19. Solutions of the example problems with the technique of Kottas and Lau, dynamic programming procedure and improvement procedure for $RAN_2 = 2L$

Example No.	Kottas and Lau solution †	DP procedure solution at α_1 †	Imp.procedure solution †	percentage difference between imp.procedure solution and	
				Kottas and Lau solution	DP procedure solution
1	36.146	36.146	34.485	4.6	4.6
2	25.400	25.400	25.400	0.0	0.0
3	36.369	36.369	36.369	0.0	0.0
4	10.180	9.246	9.246	9.2	0.0
5	11.817	9.937	9.937	15.9	0.0
6	15.589	15.492	15.492	0.6	0.0
7	43.246	39.982	38.909	10.0	2.7
8	54.481	48.516	48.516	11.0	0.0
9	43.283	43.283	43.283	0.0	0.0
10	26.337	23.434	21.894	11.0	6.6
11	33.027	32.780	32.713	1.0	0.2
12	20.184	20.174	20.174	0.1	0.0
13	72.071	68.214	68.214	5.4	0.0
14	28.846	28.806	28.806	0.1	0.0
15	10.335	10.355	10.355	0.0	0.0
16	25.814	25.766	25.766	0.2	0.0
17	11.457	11.449	11.449	0.0	0.0
18	36.509	32.117	32.117	12.0	0.0
19	15.774	16.392	15.000	4.9	8.5
20	40.029	36.141	36.128	9.8	0.0
21	8.833	8.160	8.160	7.6	0.0
22	34.715	30.959	30.959	10.8	0.0
23	22.559	19.398	19.398	14.0	0.0
24	54.939	54.939	54.939	0.0	0.0
25	35.611	33.048	33.048	7.2	0.0
26	76.071	73.071	73.071	3.9	0.0
27	28.826	26.031	26.031	9.7	0.0
28	8.716	8.689	8.689	0.3	0.0
29	30.451	27.638	27.638	9.2	0.0
30	15.390	14.207	14.207	7.7	0.0

† Solution values are in $/unit

Table 20. Solutions of the example problems with the technique of Kottas and Lau, dynamic programming procedure and improvement procedure for $RAN_2 = 3L$

Example No.	Kottas and Lau solution †	DP procedure solution at α_1 †	Imp.procedure solution †	percentage difference between imp.procedure solution and	
				Kottas and Lau solution	DP procedure solution
1	41.078	41.078	41.078	0.0	0.0
2	25.458	25.458	25.458	0.0	0.0
3	37.067	37.067	37.067	0.0	0.0
4	9.966	9.711	9.711	2.6	0.0
5	12.085	10.205	10.205	15.6	0.0
6	15.667	15.521	15.521	0.9	0.0
7	46.917	46.916	46.916	0.0	0.0
8	63.920	58.927	57.429	10.2	2.5
9	44.008	44.008	44.008	0.0	0.0
10	27.921	25.028	24.217	10.4	3.2
11	33.369	32.998	32.898	1.4	0.3
12	21.384	21.369	21.369	0.0	0.0
13	83.304	79.999	79.999	4.0	0.0
14	28.866	28.806	28.806	0.2	0.0
15	11.528	11.528	11.528	0.0	0.0
16	25.838	25.766	25.766	0.3	0.0
17	11.482	11.470	11.470	0.1	0.0
18	35.823	32.152	32.152	10.3	0.0
19	16.749	16.932	15.096	9.9	10.8
20	42.490	38.608	38.589	9.2	0.0
21	8.955	8.638	8.638	3.5	0.0
22	37.229	33.716	33.716	9.4	0.0
23	23.034	20.452	20.452	11.2	0.0
24	61.794	61.794	61.794	0.0	0.0
25	40.911	38.317	38.317	6.3	0.0
26	87.040	85.001	85.001	2.3	0.0
27	29.033	26.260	26.260	9.6	0.0
28	8.887	8.846	8.846	0.5	0.0
29	32.412	30.322	30.322	6.5	0.0
30	16.354	15.253	15.253	6.7	0.0

† Solution values are in $/unit

Table 21. Solutions of the example problems with the technique of Kottas and Lau, dynamic programming procedure and improvement procedure for $RAN_2 = 4L$

Example No.	Kottas and Lau solution †	DP procedure solution at α_1 †	Imp.procedure solution †	percentage difference between imp.procedure solution and	
				Kottas and Lau solution	DP procedure solution
1	46.011	46.011	46.011	0.0	0.0
2	25.516	25.516	25.516	0.0	0.0
3	37.766	37.766	37.766	0.0	0.0
4	10.120	10.008	10.008	1.1	0.0
5	12.354	10.473	10.473	15.2	0.0
6	15.745	15.551	15.551	1.2	0.0
7	53.851	53.851	53.851	0.0	0.0
8	69.085	68.920	68.920	0.2	0.0
9	44.733	44.733	44.733	0.0	0.0
10	29.514	26.622	26.505	9.8	0.4
11	33.711	33.216	33.083	1.9	0.4
12	22.584	22.565	22.565	0.0	0.0
13	94.536	91.784	91.784	2.9	0.0
14	28.886	28.806	28.806	0.3	0.0
15	12.702	12.702	12.702	0.0	0.0
16	25.862	25.766	25.766	0.4	0.0
17	11.507	11.491	11.491	0.1	0.0
18	35.895	32.186	32.186	10.3	0.0
19	16.902	17.146	15.192	10.1	11.4
20	44.950	41.075	41.049	8.7	0.1
21	9.477	8.475	8.475	10.6	0.0
22	39.743	36.472	36.472	8.2	0.0
23	23.508	21.218	21.218	9.7	0.0
24	68.649	68.649	68.649	0.0	0.0
25	46.087	43.586	43.586	5.4	0.0
26	98.009	96.930	96.930	1.1	0.0
27	29.239	26.490	26.490	9.4	0.0
28	9.058	9.003	9.003	0.6	0.0
29	34.718	33.007	33.007	4.9	0.0
30	17.400	16.298	16.298	6.3	0.0

† Solution values are in $/unit

Table 22. 90% confidence interval limits on percentage differences between the solutions of the improvement procedure, technique of Kottas and Lau and dynamic programming procedure

RAN$_2$	90 % confidence interval limits on percentage difference between the solution of improvement procedure and	
	Kottas and Lau solution	DP procedure solution
L	[20.7 ; 24.2]	[16.3 ; 20.8]
U[L,2L]	[7.5 ; 11.7]	[2.7 ; 6.7]
2L	[4.0 ; 7.2]	[0.2 ; 1.3]
3L	[3.0 ; 5.9]	[-0.1 ; 1.2] †
4L	[2.5 ; 5.3]	[-0.3 ; 1.0] †

† The lower limit of the interval is negative, because the mean value is less than $\frac{t_{\alpha/2,n-1}\, s}{\sqrt{n}}$, although all the percentage differences are positive

4. Kottas and Lau technique and the dynamic programming procedure tend to be insensitive to the relative magnitude of the incompletion costs, whereas the improvement procedure gives solutions that are very close to the optimum ones for all relative magnitudes of incompletion costs. Furthermore, it can be inferred that both Kottas and Lau technique and the dynamic programming procedure perform better for large relative magnitudes of incompletion costs (large RAN_2 values) than at the lower values.

6.0 Extensions Of The Procedures

In this chapter, some of the assuptions of the assembly line balancing problem stated in Chapter 3 will be relaxed and the extensions of the methodology to these cases will be presented. The first assumption relaxed involves the type of task performance time distributions. The tasks can have performance times distributed according to probability distributions other than Normal distribution if certain conditions are met. The extension to other distribution functions is presented in Section 6.1. The second assumption relaxed is related to the restrictions on making station assignments. There can be restrictions other than the precedence constraints on making station assignments. The model is easily extended to cover such restrictions, and the discussion of this extension is presented in Section 6.2. In Section 6.3, the single-machine sequencing procedure presented in Section 5.3.2.1 will be extended to the case in which the cumulative incompletion costs of the tasks are proportional to their expected performance times. This can be interpreted as relaxing the precedence constraints among the tasks. The solution procedure developed for this version of the problem is shown to generate almost optimal solutions. The same extension is also made to the M-machine scheduling procedure presented in Section 5.3.3.1. The procedure developed for this version of the scheduling problem is presented in Section 6.4. A worst-case analysis on the ratio of the heuristic and optimal solutions is also presented, and a bound on the ratio is derived.

6.1 Extension To Other Performance Time Distributions

The dynamic programming procedure described in Chapter 4 and the approximation procedure presented in Chapter 5 apply to problems with tasks having performance times distributed according to probability distribution functions satisfying the following assumptions:

Assumption 1. Processing times of the tasks are independently distributed nonnegative random variables and their distributions, $F_{\theta_i}(t_i)$ belong to a class of similar distributions characterized by a single parameter, $\theta_i > 0$ for $i = 1,...,N$.

Assumption 2. If $T = \sum_{i \in A} t_i$, where A is any subset of the N tasks, then the distribution of T belongs to the same class, with $\theta_T = \sum_{i \in A} \theta_i$.

In Assumption 1, the independence and nonnegativity properties of the performance times are straightforward. The dynamic programming procedure does not require performance time distributions to be characterized by a single parameter. On the other hand, it requires that the distribution of the sum of the performance times in any subset of the tasks belongs to the same class as the individual task performance times. The sequencing and scheduling procedures developed in Chapter 5 require task performance time distributions to be characterized by a single parameter.

The Normal distribution does not, in reality, satisfy the nonnegativity and single-parameter requirements. In order to satisfy the nonnegativity requirement, it is truncated at zero. If the coefficient of variation of task performance times, a is sufficiently small, then the effect of this truncation is minimal as shown in Section 3.1. The single-parameter requirement for the sequencing and scheduling procedures of Chapter 5 is overcome by assuming $\sigma_i = a \times \mu_i$ for all i.

The task performance times can be represented by probability distributions other than the Normal distribution as stated in the following Corollary.

Corollary 6.1. The dynamic programming procedure and the improvement procedure presented in this dissertation could be easily extended to problems with tasks having performance times distributed according to Poisson, Gamma, Binomial, Negative Binomial and Chi-square distributions.

The Chi-square and Poisson distributions are closed under convolution. That is, the sum of any number of performance times is distributed according to that of the individual task performance times. The other distributions stated in the Corollary above are closed under convolution when certain conditions are met. For Gamma distribution, if $X_i \sim \text{Gamma}(\alpha_i, \beta)$ for $i = 1,...,N$, then $X_1 + ... + X_j \sim \text{Gamma}(\alpha_1 + ... + \alpha_j, \beta)$, where α and β are the shape and scale parameters of the distribution, respectively. In other words, task performance time distributions can be extended to Gamma distribution if the scale parameter of the distribution is the same for all the tasks. For Binomial distribution, if $X_i \sim \text{Bin}(t_i, p)$ for $i = 1,...,N$, then $X_1 + ... + X_j \sim \text{Bin}(t_1 + ... + t_j, p)$. For Negative Binomial distribution, if $X_i \sim \text{Nbin}(s_i, p)$ for $i = 1,...,N$, then $X_1 + ... + X_j \sim \text{Nbin}(s_1 + ... + s_j, p)$. Therefore, task performance time distributions can be extended to Binomial and Negative Binomial distributions if the p parameter of the distributions is the same for all the tasks.

The choice of the task performance time distribution should be made with great care, since it affects the output line design significantly. Task performance times should be examined carefully so that the chosen probability distribution function represents them most accurately.

In the next section the extension of the model to station assignment restrictions other than the precedence constraints will be discussed.

6.2 Extension To Other Station Assignment Restrictions

One of the assumptions of the assembly line balancing problem as stated in Chapter 3 is that the only restrictions on assigning tasks to stations are the precedence constraints. On the other hand, two other types of restrictions are quite common in industry; these are the restrictions imposed by fixed facilities or machines on the line and the restrictions of position. The procedures developed in this dissertation can easily be extended to cover these restrictions.

Fixed facility restrictions are caused by the immovable stations on the line. These facilities may be machines, processes, testing facilities or indexing apparatus.

Positional restrictions can be classified as front-and-back and top-and-bottom restrictions. The front-and-back restrictions are caused by the work that must be done either on the front or back of the item. Since it is not desirable to cross over the assembly line, these tasks are separated from each other. Top-and-bottom restrictions exist where the item is either inverted or elevated above the worker's head. It becomes more economical to gather the tasks to be done on the underside of the item into the same stations.

In the dynamic programming procedure described in Chapter 4, these additional restrictions are considered when the decision variables corresponding to a state are generated. The decision variables violating these restrictions are not generated. In the improvement procedure, the node generation process described in Section 5.3.2 considers these restrictions by not placing the tasks in the unmarked immediate follower list if the tasks violate the restrictions.

6.3 Extension Of The Single-Machine Sequencing Procedure To Independent Tasks

The single-machine sequencing procedure presented in Section 5.3.2.2 applies to problems in which the cumulative incompletion costs of the tasks are independent of their expected performance times. It can be interpreted as having precedence constraints among the tasks, so the cumulative incompletion costs depend on the positions of the tasks on the precedence diagram.

In this section, we assume that the cumulative incompletion costs of the tasks are proportional to their expected performance times. In other words, we relax the precedence constraints among the tasks. A procedure will be developed to solve this version of the problem. In the sequel, we first discuss the main result which specifies an underlying property of the so called promising sequences. A procedure to generate such sequences is then presented. Also, some computational experience is reported.

6.3.1 Development Of The Single-Machine Sequencing Procedure With Independent Tasks

The notation and assumptions of this version of the problem are the same as the case in which precedence constraints among the tasks exist as presented in Section 5.3.2.2. The objective function of this version of the problem is as follows:

$$\underset{s \in S}{\mathrm{Min}} \sum_{i=1}^{N} IC_i \times \Pr\{ f_i(s) > C \}$$

The standard deviation of a task performance time was assumed to be linearly proportional to its expected performance time as discussed in Section 5.3.2. In this section, we assume that the variance of a task performance time is linearly proportional to its expected performance time. In other words, $\sigma_i^2 = a \times \mu_i$ for all i. The effect of this new assumption on the truncation of task performance time distributions at zero is analyzed below. The analysis indicates that the effect of this new assumption on the truncation can also be ignored. By considering this new assumption, the analysis on the truncation of performance time distibutions gets completed.

Task performance time distributions are truncated at zero. This assumption of truncating performance time distributions at zero can be made if the probability that a normally distributed random variable can take negative values is small enough as discussed in Section 3.1. Next, we develop some conditions under which this is true. To that end, let E represent the area to the left of zero under a Normal distribution with mean μ_i and variance σ_i^2. If ε is a small quantity greater than zero, then the desired condition is as follows:

$$E = \Phi\left(\frac{-\mu_i}{\sigma_i}\right) \leq \varepsilon \qquad \text{for } i = 1,.....,N$$

The above condition reduces to the following expression:

$$\mu_i \geq a \times [\,\Phi^{-1}(\varepsilon)\,]^2 \qquad \text{for } i = 1,....,N$$

In other words, the truncation of the Normal distribution can be ignored if the expected performance times of the tasks are larger than the above value determined as a function of ε and a. Table 23 depicts the lower bounds on the expected performance times for different ε and a values. Note that for practical task performance times, $a \leq 1.0$, since $a > 1$ implies that the variance of a task performance time is greater than its expected value and this situation is highly improbable for tasks encountered in industry.

Table 23. Lower bounds on the expected performance times of the tasks

ε	$\mu_i \geq$	Lower bound on μ_i for all i				
		$a = 0.1$	$a = 0.2$	$a = 0.3$	$a = 0.5$	$a = 1.0$
0.20	0.706 a	0.071	0.141	0.212	0.353	0.706
0.10	1.638 a	0.164	0.328	0.491	0.819	1.638
0.05	2.706 a	0.271	0.541	0.812	1.353	2.706
0.03	3.346 a	0.353	0.707	1.060	1.767	3.534
0.01	5.406 a	0.541	1.081	1.622	2.037	5.406

The ε value column of Table 23 gives the area under the Normal distribution function to the left of zero that is discarded. For example, for $\varepsilon = 0.05$, we assume that when the area under the Normal distribution function to the left of zero is equal to or less than 0.05, it is negligible. The second column of the Table depicts the lower bounds on μ_i for all i as functions of a. In other words, μ_i should be larger than the values given in the column for all i in order to make the area under the Normal distribution function to the left of zero less than or equal to the corresponding ε value. The other columns of the Table depict the lower bounds on μ_i for all i for different a values. For example, for $a = 0.2$, if $\mu_i \geq 0.541$ for all i, then the area under the Normal distribution function to the left of zero is less than or equal to 0.05. Based on this discussion the Normal distribution of task performance times can safely be assumed to be truncated at zero for practical problems.

The incompletion probability function, p(x) is monotonically increasing and convex over the interval $0 \leq x \leq C'$ and monotonically increasing and concave for $x \geq C'$, for some $C' < C$. This property of the incompletion probability function will be used in the development of the sequencing rule and is formally stated and proved next.

Theorem 6.1. Incompletion probability function, p(x) is monotonically increasing and convex over the interval $0 \leq x \leq C'$ and monotonically increasing and concave for $x \geq C'$, where $C' < C$.

Proof. Incompletion probability function, p(x), can be represented as follows:

$$p(x) = 1 - \Phi\left(\frac{C - x}{\sqrt{ax}}\right) = \Phi\left(\frac{x - C}{\sqrt{ax}}\right) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{b(x)} e^{-z^2/2} dz$$

where $b(x) = \frac{x - C}{\sqrt{ax}}$. Let $f(z) = e^{-z^2/2}$. p(x) is monotonically increasing and convex over an interval if $\frac{dp(x)}{dx} > 0$ and $\frac{d^2p(x)}{dx^2} > 0$ over that interval, and is monotonically increasing and concave if $\frac{dp(x)}{dx} > 0$ and $\frac{d^2p(x)}{dx^2} < 0$ over that interval. In order to show this, consider $\frac{dp(x)}{dx}$ first.

$$\frac{dp(x)}{dx} = \frac{1}{\sqrt{2\pi}} f(b) \frac{db}{dx} = \frac{1}{\sqrt{2\pi}} e^{-b^2/2} \frac{(x + C)}{2\sqrt{a}\, x^{3/2}}$$

Hence, $\frac{dp(x)}{dx} > 0$ for $x > 0$. Next, consider $\frac{d^2p(x)}{dx^2}$.

$$\frac{d^2p(x)}{dx^2} = \frac{1}{\sqrt{2\pi}} \left[f'(b) \left(\frac{db}{dx}\right)^2 + f(b) \frac{d^2b}{dx^2} \right] \qquad [6.1]$$

$$\text{where } f'(b) = -b e^{-b^2/2} = -b f(b) \qquad [6.2]$$

Substituting Equation [6.2] into Equation [6.1] yields:

$$\frac{d^2p(x)}{dx^2} = \frac{f(b)}{\sqrt{2\pi}} \left[\frac{d^2b}{dx^2} - b \left(\frac{db}{dx}\right)^2 \right] \qquad [6.3]$$

$$\text{Moreover, } \left(\frac{db}{dx}\right)^2 = \frac{(x + C)^2}{4 a x^3} \qquad [6.4]$$

$$\text{and } \frac{d^2b}{dx^2} = -\frac{x + 3C}{4\sqrt{a}\, x^{5/2}} \qquad [6.5]$$

Therefore, substituting Equations [6.4] and [6.5] into Equation [6.3] yields:

$$\frac{d^2p(x)}{dx^2} = -\frac{e^{-b^2/2}}{\sqrt{2\pi}} \left[\frac{ax(x + 3C) + (x - C)(x + C)^2}{4 a^{3/2} x^{7/2}} \right]$$

Let $\Delta(x) = ax(x + 3C) + (x - C)(x + C)^2$. Note that the signs of $\frac{d^2p(x)}{dx^2}$ and $\Delta(x)$ are opposite of each other. To determine the nature of $\Delta(x)$, consider $\frac{d\Delta(x)}{dx} = 3x^2 + 2x(a + C) + 3aC - C^2$. As x approaches zero, $\frac{d\Delta(x)}{dx} \geq 0.0$ if $a \geq \frac{C}{3}$ and $\frac{d\Delta(x)}{dx} \leq 0.0$ if $a \leq \frac{C}{3}$. On the other hand, for $x \geq C$, $\frac{d\Delta(x)}{dx} \geq 0.0$. Thus, for the case $a \leq \frac{C}{3}$, the slope of $\Delta(x)$ changes sign as x moves from C to zero. For this case, let $0.0 \leq y^* \leq C$ be such that $\frac{d\Delta(y^*)}{dx} = 0.0$. Since $\frac{d^2\Delta(y^*)}{dx^2} = 6y^* + 2(a + C) \geq 0.0$, then y^* is a local minimum. In addition, for $x = 0.0$, $\Delta(x) < 0$ and for $x \geq C$, $\Delta(x) > 0$; therefore, it follows that $\Delta(x) \leq 0$ over the interval $0 \leq x \leq C'$, where $C' \leq C$ and $\Delta(x) \geq 0$ for $x \geq C'$. $\Delta(x)$ is depicted in Figure 16 for the case when $a \geq \frac{C}{3}$, and in Figure 17 for the case when $a \leq \frac{C}{3}$. Consequently, $\frac{d^2p(x)}{dx^2} \geq 0.0$ for $0 \leq x \leq C'$, and $\frac{d^2p(x)}{dx^2} \leq 0.0$ for $x \geq C'$. This proves that the incompletion probability function, p(x) is monotonically increasing and convex over the interval $0 \leq x \leq C'$, and monotonically increasing and concave for $x \geq C'$, where $C' < C$. Note that C' is the root of $\Delta(x)$. In order to determine C', let $r = -\frac{(a + C)^2}{3} + 3aC - C^2$ and $q = 2[\frac{a + C}{3}]^3 - \frac{(a + C)(3aC - C^2)}{3} - C^3$. If $V = [\frac{r}{3}]^3 + [\frac{q}{2}]^2$, then,

$$C' = [\ -\frac{q}{2} + \sqrt{V}\,]^{1/3} + [\ -\frac{q}{2} - \sqrt{V}\,]^{1/3} - \frac{(a + C)}{3}$$

This completes the proof.♦

The derivation of C' gets quite complicated; but when the values of C and a are known, the computation becomes quite straightforward. We computed C' values for different C and a values. Table 24 depicts C' values for the values of C in the range from 1 to 20, and for a values of 0.2, 0.5 and 1.0. The ratios of C' and C are also depicted in the Table. As it is seen, for $C \geq 10$, C' gets quite close to C. In addition, the ratio of C' and C is inversely proportional to the value of a; in fact, as a approaches zero, the difference between C and C' goes to zero. Thus, the value of $a = 1.0$ results in the smallest $\frac{C'}{C}$ values while those for $a = 0.2$ result in the largest values; by assumption $a \leq 1.0$. Hence, based on the above analysis, the difference between C and C' for practical problem parameters with $C > 10$ and $a \leq 0.5$ can be ignored; the error involved will be negligible.

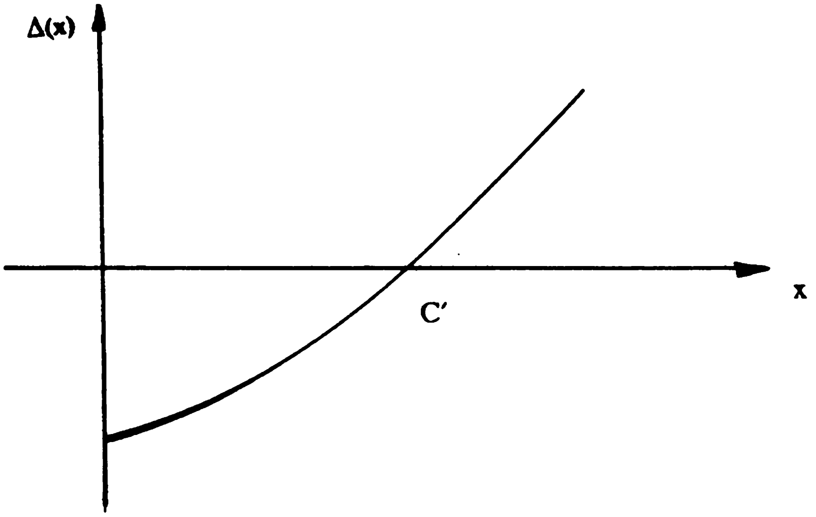


Figure 16. $\Delta(x)$ for $a \geq C / 3$

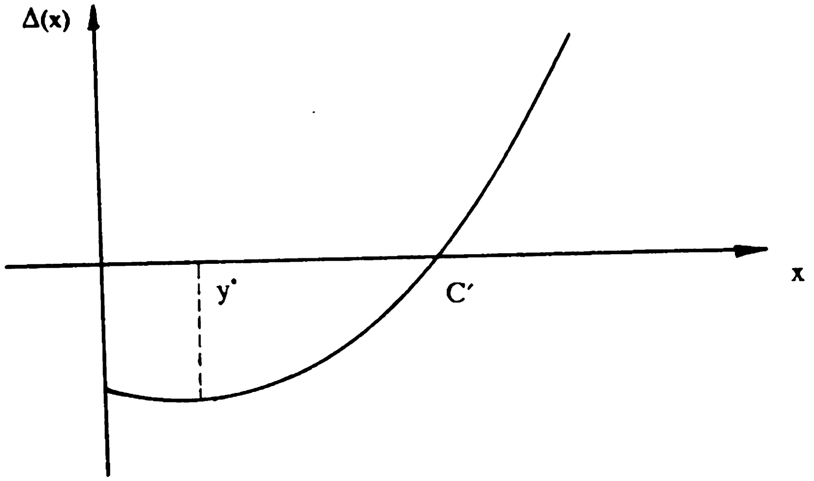


Figure 17. $\Delta(x)$ for $a \leq C / 3$

Table 24. Variation in the ratio of C′ and C for different a and C values

a	C	C′	C′ / C
0.2	1.0	0.815	0.815
	2.0	1.806	0.903
	3.0	2.804	0.935
	4.0	3.802	0.951
	5.0	4.802	0.960
	10.0	9.801	0.980
	15.0	14.801	0.987
	20.0	19.800	0.990
0.5	1.0	0.585	0.585
	2.0	1.536	0.768
	3.0	2.524	0.841
	4.0	3.517	0.879
	5.0	4.513	0.903
	10.0	9.507	0.951
	15.0	14.505	0.967
	20.0	19.503	0.975
1.0	1.0	0.352	0.352
	2.0	1.167	0.584
	3.0	2.103	0.701
	4.0	3.074	0.769
	5.0	4.057	0.811
	10.0	9.027	0.903
	15.0	14.018	0.925
	20.0	19.013	0.951

To highlight the difference between C and C′, Figure 18 depicts the ratio of C′ and C for different a and C values. Note that for $C \geq 10.0$, the ratio of C′ and C is almost equal to unity for all a values. As a approaches to zero, the ratio of C′ and C approaches to unity. Figure 19 depicts the incompletion probability functions for $a = 0.2$, $a = 1.0$ and $a = 10.0$ when $C = 1$. C is chosen to be small so that the effect of the difference between C′ and C on the shape of the incompletion probability function is highlighted. As a increases, the inflection point (C′) of the curve moves to left. In fact, as a goes to infinity, the inflection point disappears and the incompletion probability function then becomes monotonically increasing and concave over all x. Note that for $a = 0.2$, $C' = 0.815$ and for $a = 1.0$, $C' = 0.352$ as indicated in Table 24.

Figure 20 depicts a portion of the incompletion probability function and shows the incompletion probabilities of two tasks, i and j, assigned to the same position in a sequence and the incompletion probability of the later task when they are assigned one after each other. The figure assumes that $\mu_Z + \mu_i + \mu_j \leq C'$, so that this portion of the function is monotonically increasing and convex. The contribution of task i to the incompletion probability, when assigned to position μ_Z, is designated in the Figure by A, whereas its contribution is designated by B when it is assigned to position $\mu_Z + \mu_j$. Clearly, $B > A$ as the function is monotonically increasing and convex. Hence, it follows that,

$$\text{If } \mu_Z + \mu_i + \mu_j \leq C', \text{ then } p_Z + (p_i - p_Z) + (p_j - p_Z) \leq p .$$

Similarly, referring to Figure 21, it can be shown that

$$\text{If } \mu_Z \geq C', \text{ then } p_Z + (p_i - p_Z) + (p_j - p_Z) \geq p .$$

because, the incompletion probability function is monotonically increasing and concave in this region.

Furthermore, the incompletion probability of task i can also be represented as $p_i = p_{Z_i} + k_{i,\mu_{Z_i}} IC_i$ where p_{Z_i} is the incompletion probability of the task preceding task i in the

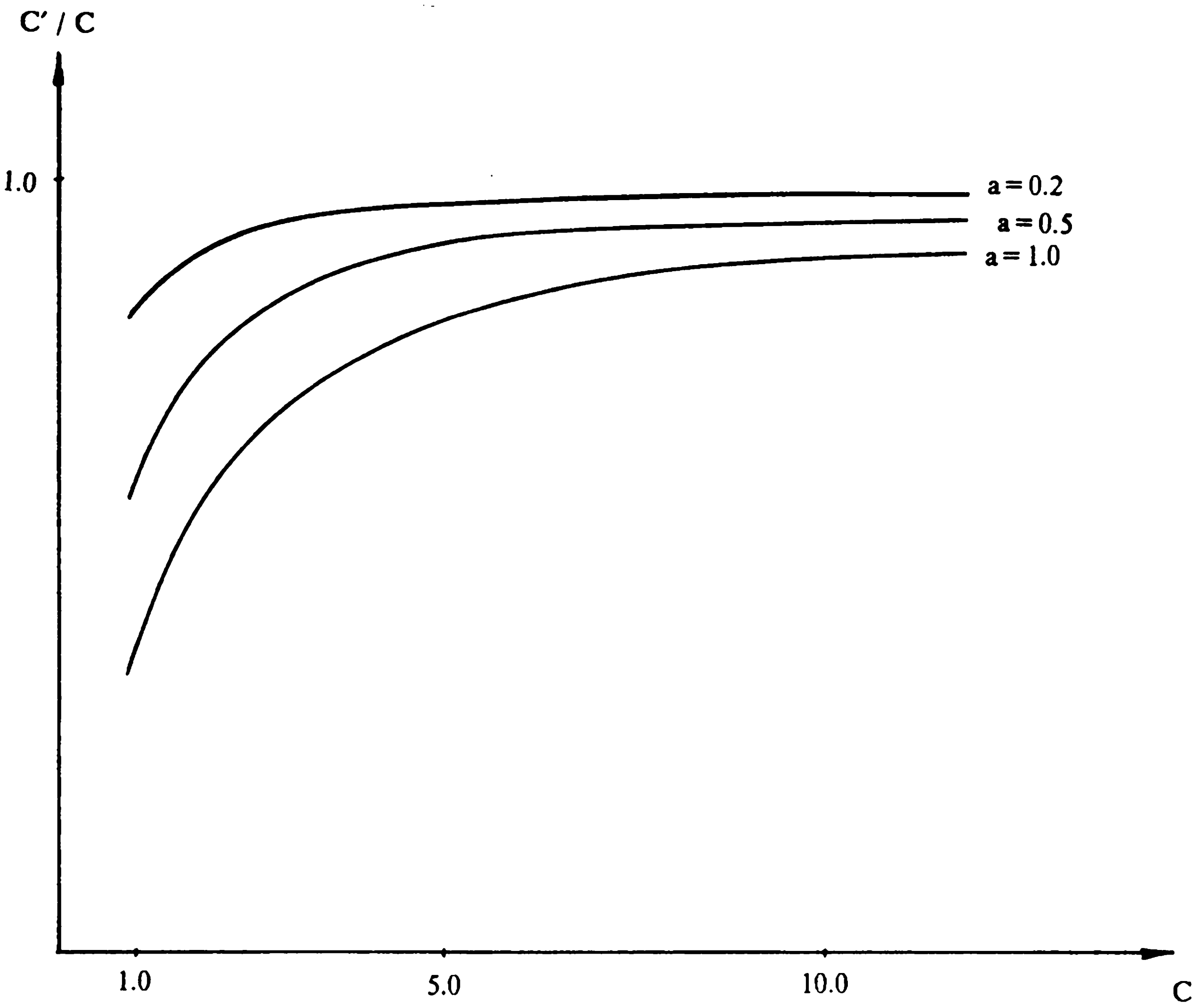


Figure 18. Ratio of C' and C for different a and C values

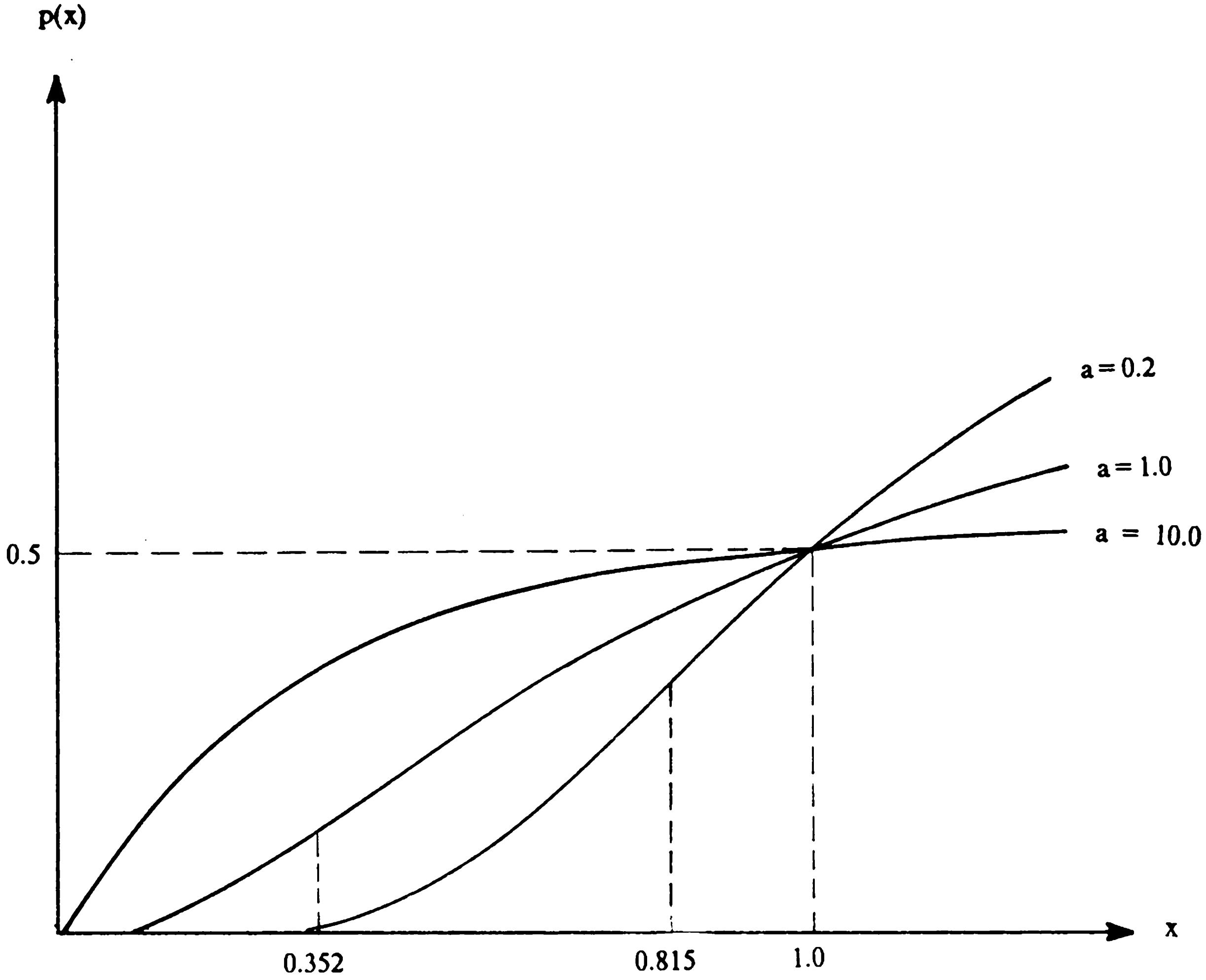


Figure 19. Incompletion probability functions for a = 0.2, a = 1.0 and a = 10.0 when C = 1.0 and the associated inflection points

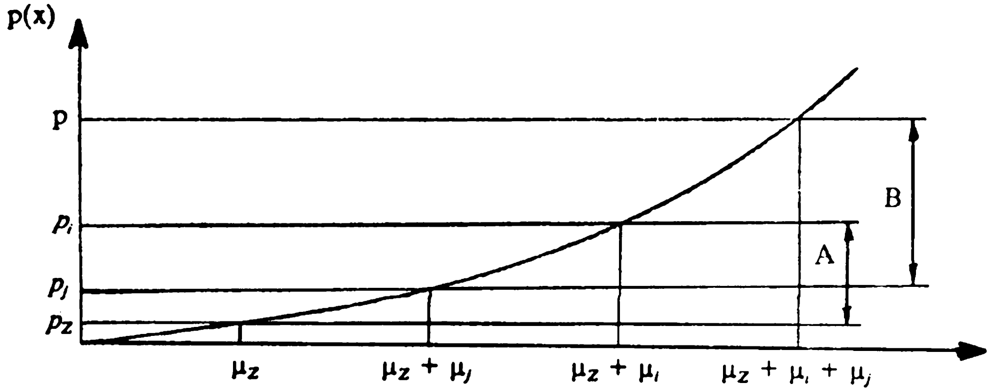


Figure 20. Incompletion probability function for the case when $x \leq C'$

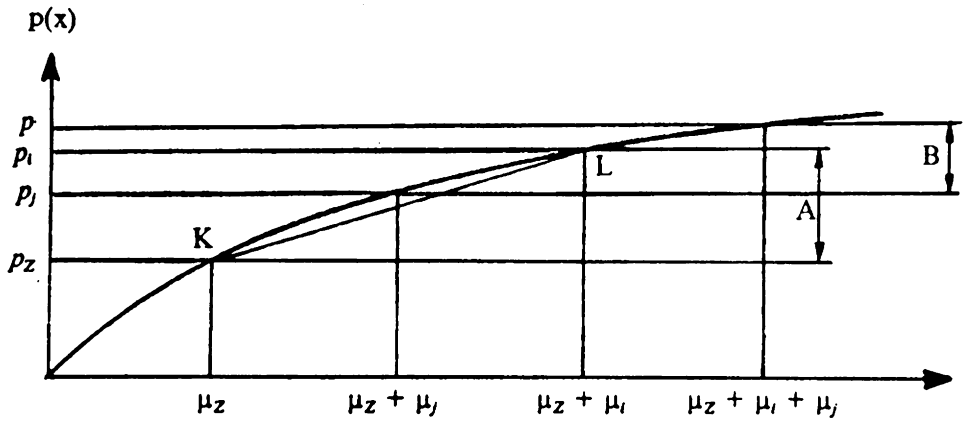


Figure 21. Incompletion probability function for the case when $x \geq C'$

sequence and μ_{Z_i} is the sum of the means of the tasks preceding task i, and $k_{i,\mu_{Z_i}}$ is a constant that depends on the position of task i in the sequence, as reflected by μ_{Z_i}, and the expected processing time of task i. Note that p_i represents p(x) when $x = \mu_{Z_i} + \mu_i$ which is the expected completion time of task i. Referring to Figure 21, $k_{i,\mu_{Z_i}}$, in reality, represents the slope of the line drawn from $(\mu_Z ; p_Z)$ to $(\mu_Z + \mu_i ; p_i)$. These points are marked as K and L in Figure 21 and the line that represents the slope $k_{i,\mu_{Z_i}}$ is also shown.

The following observation can be made due to the monotonically increasing and convex shape of the incompletion probability function for the case when $\mu_Z + \mu_i \leq C'$:

If μ_{Z} [illegible] $\mu_i \leq C'$ and $\mu_{Z_i} = \mu_{Z_j}$ and $IC_i \geq$ I[illegible]hat is, $\mu_i \geq \mu_j$) , then $k_{i,\mu_{Z_i}} \geq k_{j,\mu_{Z_j}}$

Similarly, for the monotonically increasing and concave portion of the incompletion probability function, we have,

If $\mu_{Z_i} = \mu_{Z_j} \geq C'$ and $IC_i \geq IC_j$ (that is, $\mu_i \geq \mu_j$) , then $k_{i,\mu_{Z_i}} \leq k_{j,\mu_{Z_j}}$

The following Theorem gives the criteria to determine the relative order in which two tasks should appear in the optimal sequence. Note that in sequence R, task j follows task i, whereas in sequence R′, task i follows task j.

Theorem 6.2. If $\mu_Z + \mu_i + \mu_j \leq C'$ and $IC_i \geq IC_j$, then Cost(R) ≤ Cost(R′) and if $\mu_Z \geq C'$ and $IC_i \geq IC_j$, then Cost(R) ≥ Cost(R′) .

Proof. The proof of the Theorem has two parts:

1. To show that if $\mu_Z + \mu_i + \mu_j \leq C'$ and $IC_i \geq IC_j$, then Cost(R) ≤ Cost(R′)

For this case, from above, $p_Z + (p_i - p_Z) + (p_j - p_Z) \leq p$ [6.6]

A quantity, S can be added to the left side of Equation [6.6] without violating it as long as $S \leq p + p_Z - p_i - p_j$. Let such an S be,

$$S = \left[p_Z - p_i - p_j + \frac{IC_i\, p_i - IC_j\, p_j}{IC_i - IC_j} \right]$$

We, first, show that this $S \le p + p_Z - p_i - p_j$. The proof is by deduction. If, after substituting for S,

$$p_Z - p_i - p_j + \frac{IC_i\, p_i - IC_j\, p_j}{IC_i - IC_j} \le p + p_Z - p_i - p_j$$

then, $IC_j\,(p - p_j) \le IC_i\,(p - p_i)$ (after simplification) [6.7]

Since $p - p_j = k_{i,\mu_{Z_i}+\mu_j} IC_i$ and $p - p_i = k_{j,\mu_{Z_j}+\mu_i} IC_j$, then Equation [6.7] becomes:

$IC_j\,(k_{i,\mu_{Z_i}+\mu_j} IC_i) \le IC_i\,(k_{j,\mu_{Z_j}+\mu_i} IC_j)$ as $k_{i,\mu_{Z_i}+\mu_j} \le k_{j,\mu_{Z_j}+\mu_i}$ and $IC_i,\ IC_j \ge 0.0$

But, this is true, since $k_{i,\mu_{Z_i}+\mu_j}$ and $k_{j,\mu_{Z_j}+\mu_i}$ represents the slopes of the lines drawn from $(\mu_Z + \mu_j\,;\ p_j)$ and $(\mu_Z + \mu_i\,;\ p_i)$ to $(\mu_Z + \mu_i + \mu_j\,;\ p)$, respectively, on the incompletion probability curve (Figure 20), and $\mu_i \ge \mu_j$, because $IC_i \ge IC_j$.

Now, by adding S to the left side of Equation [6.6], it becomes:

$$p_Z + (p_i - p_Z) + (p_j - p_Z) + (p_Z - p_i - p_j) + \frac{IC_i\, p_i - IC_j\, p_j}{IC_i - IC_j} \le p \qquad [6.8]$$

Since $p_i - p_Z = k_{i,\mu_{Z_i}} IC_i$ and $p_j - p_Z = k_{j,\mu_{Z_j}} IC_j$, then Equation [6.8] becomes:

$$p_Z + k_{i,\mu_{Z_i}} IC_i + k_{j,\mu_{Z_j}} IC_j + (p_Z - p_i - p_j) + \frac{IC_i\, p_i - IC_j\, p_j}{IC_i - IC_j} \le p$$

$$p_Z(IC_i - IC_j) + (k_{i,\mu_{Z_i}} IC_i + k_{j,\mu_{Z_j}} IC_j)(IC_i - IC_j) + (p_Z - p_i - p_j)(IC_i - IC_j) +$$

$$+ \quad {}_i\, p_i - IC_j\, p_j) \le p(IC_i - IC_j) \qquad [6.9]$$

The terms $(p_Z - p_i - p_j)(IC_i - IC_j) + (IC_i\,p_i - IC_j\,p_j)$ of the above expression can be further reduced as follows:

$$(p_Z - p_i - p_j)(IC_i - IC_j) + (IC_i\,p_i - IC_j\,p_j) = IC_iIC_j\left[\frac{p_i - p_Z}{IC_i} - \frac{p_j - p_Z}{IC_j}\right]$$

$$= IC_iIC_j\left[k_{i,\mu_{Z_i}} - k_{j,\mu_{Z_j}}\right] \qquad [6.10]$$

Substituting Equation [6.10] into Equation [6.9] yields:

$$p_Z(IC_i - IC_j) + (k_{i,\mu_{Z_i}}IC_i + k_{j,\mu_{Z_j}}IC_j)(IC_i - IC_j) + IC_iIC_j(k_{i,\mu_{Z_i}} - k_{j,\mu_{Z_j}}) \le p\,(IC_i - IC_j)$$

$$p_Z(IC_i - IC_j) + k_{i,\mu_{Z_i}}IC_i^2 - k_{j,\mu_{Z_j}}IC_j^2 \le p\,(IC_i - IC_j)$$

$$p_i\,IC_i + p\,IC_j \le p_j\,IC_j + p\,IC_i$$

That is, $\mathrm{Cost}(R) \le \mathrm{Cost}(R')$

2. To show that if $\mu_Z \ge C'$ and $IC_i \ge IC_j$, then $\mathrm{Cost}(R) \ge \mathrm{Cost}(R')$

Again for this case, from above, $p_Z + (p_i - p_Z) + (p_j - p_Z) \ge p$ [6.11]

A quantity S can be added to the left side of Equation [6.11] without violating it as long as $S \ge p + p_Z - p_i - p_j$ (Note that $p + p_Z - p_i - p_j \le 0.0$).

We, first, show that $S \ge p + p_Z - p_i - p_j$. After substituting for S,

$$p_Z - p_i - p_j + \frac{IC_i\,p_i - IC_j\,p_j}{IC_i - IC_j} \ge p + p_Z - p_i - p_j$$

then, $IC_j\,(p - p_j) \ge IC_i\,(p - p_i)$ (after simplification) [6.12]

Since $p - p_j = k_{i,\mu_{Z_i}+\mu_j}\,IC_i$ and $p - p_i = k_{j,\mu_{Z_j}+\mu_i}\,IC_j$, then Equation [6.12] becomes:

$$IC_j\,(k_{i,\mu_{Z_i}} + \mu_j\,IC_i) \geq IC_i\,(k_{j,\mu_{Z_j}} + \mu_i\,IC_j) \text{ as } k_{i,\mu_{Z_i}} + \mu_j \geq k_{j,\mu_{Z_j}} + \mu_i \text{ and } IC_i,\ IC_j \geq 0.0$$

But, this is true, since $k_{i,\mu_{Z_j}} + \mu_i$ and $k_{j,\mu_{Z_j}} + \mu_j$ represents the slopes of the lines drawn from $(\mu_Z + \mu_j\,;\ p_j)$ and $(\mu_Z + \mu_i\,;\ p_i)$ to $(\mu_Z + \mu_i + \mu_j\,;\ p)$, respectively, on the incompletion probability curve (Figure 21), and $\mu_i \geq \mu_j$, because $IC_i \geq IC_j$.

Now, by adding S to the left side of Equation [6.11], it becomes:

$$p_Z + (p_i - p_Z) + (p_j - p_Z) + (p_Z - p_i - p_j) + \frac{IC_i\,p_i - IC_j\,p_j}{IC_i - IC_j} \geq p \qquad [6.13]$$

Equation [6.13] can be written as:

$$p_Z\,(IC_i - IC_j) + (k_{i,\mu_{Z_i}}IC_i + k_{j,\mu_{Z_j}}IC_j)(IC_i - IC_j) + (p_Z - p_i - p_j)(IC_i - IC_j) +$$

$$+ (IC_i\,p_i - IC_j\,p_j) \geq p\,(IC_i - IC_j)$$

$$p_Z\,(IC_i - IC_j) + (k_{i,\mu_{Z_i}}IC_i + k_{j,\mu_{Z_j}}IC_j)(IC_i - IC_j) + IC_iIC_j\,(k_{i,\mu_{Z_i}} - k_{j,\mu_{Z_j}}) \geq p\,(IC_i - IC_j)$$

$$p_Z\,(IC_i - IC_j) + k_{i,\mu_{Z_i}}IC_i^2 - k_{j,\mu_{Z_j}}IC_j^2 \geq p\,(IC_i - IC_j)$$

$$p_i\,IC_i + p\,IC_j \geq p_j\,IC_j + p\,IC_i$$

That is, Cost(R) ≥ Cost(R') . ♦

In words, the above result implies the following:

1. If for any pair of tasks i and j, the sum of the expected processing times of the remaining tasks (μ_Z) is larger than or equal to C', then the sequence which minimizes expected incompletion cost has two regions.

2. The tasks are ordered in a descending order of their incompletion costs in the first region while they are ordered in an ascending order of their incompletion costs in the second region.

Note that the first item implies only a sufficient condition for two regions to exist in the optimal sequence. There could exist two regions which do not satisfy this condition. In particular, for the case when $\mu_Z + \mu_i + \mu_j \geq C'$ and $\mu_Z \leq C'$, which is not covered by the Theorem above, tasks i and j can both belong to the same or two different regions, consequently giving rise to the formation of one or two regions.

This result helps to tremendously cut down the number of sequences that need to be considered. Such sequences will hereafter be called 'promising sequences'. Theoretically speaking, given N tasks $(2^N - 1)$ different sets of tasks could be allocated in the first region. Since a second region corresponding to each first region is formed accordingly, $(2^N - 1)$ different sequences could be formed. But, only a small number of these $(2^N - 1)$ sequences are promising sequences, since for a subset of N tasks to be qualified as a first region, the sum of the expected processing times of the tasks in the subset should be less than or equal to C'. This condition is quite severe in reducing the number of promising sequences. In the next section, a procedure to generate the promising sequences is described followed by a numerical example, and then some computational results are presented.

6.3.1.1 Procedure To Generate Promising Sequences

The proposed procedure to generate the promising sequences is a special enumeration tree whose nodes represent arrangements of tasks in the first region only. Tasks in the corresponding second regions follow trivially. The steps of the procedure are as follows:

Step 1. Initialization step. Order the tasks in the descending order of their incompletion costs. Let the first task in the sequence be numbered 1, the second task as 2, and so on.

Step 2. Check of the trivial case. If the sum of the expected processing times of the tasks is less than or equal to C′, then the order obtained in the initialization step is optimal.

Step 3. Branching step. If the current first region has task e as its last task, then N-e branches are emanated from that node. Let $S_c = \{J_{[1]}, J_{[2]}, ..., J_{[a]}\}$ be the current first region, where $J_{[a]} = e$, and let S_i be the first region generated by the ith branch for i = 1,...,N-e, then

$$S_i = \{J_{[1]}, J_{[2]}, ..., J_{[a]}, e + i\} \qquad \text{for } i = 1, ..., N - e$$

After forming the first region (designated by the node), the second region is formed by ordering the remaining tasks in the ascending order of their incompletion costs and this sequence is examined in Step 4 to determine if the current node is pruned or further branched from. Note that the sequence that is used in the evaluation of the current node corresponds to node S_{N-e} . The last task of the first region and the first and the second tasks of the second region of this sequence are referred to as e, f and g, respectively, in the following steps of the procedure.

Step 4. Evaluation and pruning step. The nodes generated in Step 3 are either pruned, evaluated and branched from, or branched from without evaluation depending on the following conditions:

> **Condition 1.** The node is pruned without evaluation if $\mu_{Z_e} \geq C'$ in the complete sequence corresponding to the node.
>
> **Condition 2.** The sequence corresponding to the node is not evaluated but is branched from if $\mu_{Z_f} + \mu_f + \mu_g \leq C'$. This is explained below.
>
> **Condition 3.** If the last task of the first region is task N, then the sequence is evaluated and pruned.
>
> **Condition 4.** All other sequences are evaluated.

Steps 3 and 4 are repeated until all the nodes are pruned. This procedure enumerates all the promising sequences and determines the optimal one.

Condition 1 of Step 4 of the procedure implies that tasks e and f, $IC_e \geq IC_f$, are ordered as task f following task e. But, since $\mu_{Z_e} \geq C'$, the sequence violates Theorem 6.2, because if $\mu_{Z_e} \geq C'$ and $IC_e \geq IC_f$, then task e should follow task f for it to qualify as a promising sequence. In addition, Theorem 6.2 is violated for all the other nodes S_i, $i = 1,...,N\text{-}e\text{-}1$, branched from this node (according to Step 3 of the procedure) as well, because $\mu_{Z_e} \geq C'$ and $IC_{e+i} \geq IC_f$ for all $i = 1,...,N\text{-}e\text{-}1$. Therefore, the current node, S_c, is pruned without evaluation. Condition 2 of Step 4 implies that $\mu_{Z_f} + \mu_f + \mu_g \leq C'$, but $IC_g \geq IC_f$, and task g follows task f. This violates Theorem 6.2. Hence, node S_{N-e} can be pruned, because the sequence generated by such a node cannot be a promising sequence. On the other hand, branching from such a node to nodes S_i, $i = 1,....,N\text{-}e\text{-}1$, may generate sequences that satisfy Theorem 6.2. Hence, the current node, S_c, is not evaluated but is further branched from.

Next, we illustrate this procedure on an example problem. This example problem consists of 6 tasks. The other relevant data are shown in Table 25. Let $C' = 10$ minutes. The enumeration tree is shown in Figure 22. In the tree, the status of a branch is indicated by the condition number of Step 4 of the generating procedure. The branches that are pruned without evaluation are labelled "1" and those that are not evaluated but branched from are labelled "2". The branches that are evaluated and pruned because of the last task of the first region being task N are labelled "3". All the other branches that are evaluated and branched from are labelled "4". For example, node {1} generates the sequence {1-6-5-4-3-2} which is a promising sequence; so it is evaluated and labelled "4". On the other hand, node {2-3-4} generates the sequence {2-3-4-6-5-1} which is pruned without evaluation and labelled "1" since it satisfies condition 1 of the evaluation and pruning step.

In the example above, it can be assumed that $C = 10$ minutes and the difference between C' and C is negligible. As a matter of fact, when the difference between C' and C is not ignored, the computational requirements of the procedure decreases. In other words, when $C' \leq C$, the number of promising sequences generated (number of nodes in the tree) decreases. Table 26 depicts the C' values corresponding to different a values when $C = 10$ and the number of promising sequences generated in the tree for each C′ value. Note that when C′ goes to zero, the problem becomes trivial to solve: Tasks are sequenced in the increasing order of their incompletion costs.

Table 25. Parameters of the example problem

Task (i)	Mean (μ_i) †	Incompletion cost (IC_i) ‡
1	10	5
2	8	4
3	6	3
4	4	2
5	2	1
6	1	0.5

† Mean values are in minutes
‡ Cost values are in $

Table 26. Number of promising sequences for different a values

a	C′	Number of nodes generated
a → 0	10.000	31
0.2	9.801	31
0.5	9.507	31
1.0	9.027	31
1.5	8.563	31
2.0	8.115	31
2.5	7.685	25
3.0	7.274	25
5.0	5.832	22
10.0	3.532	21
a → ∞	0.000	0

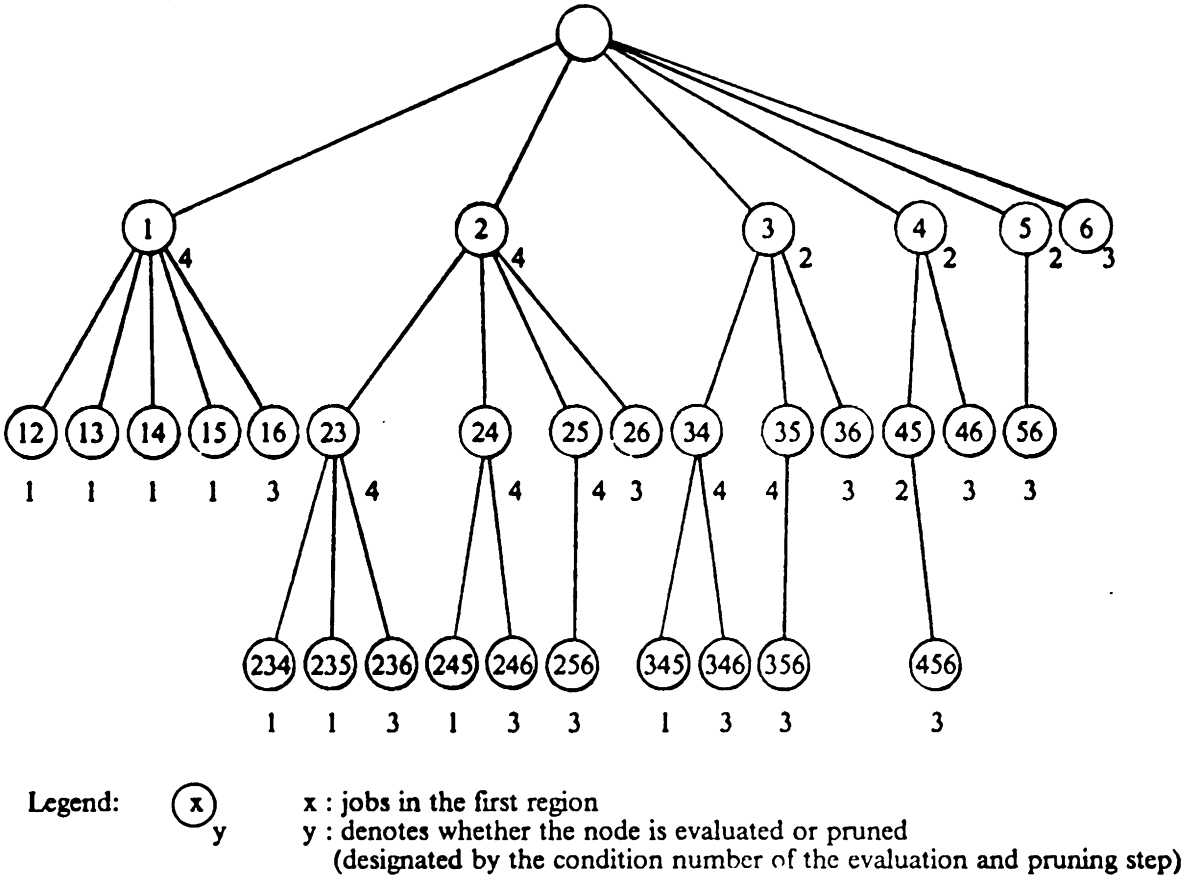

1
4
2
4
3
2
4
2
5
2
6
3
12
1
13
1
14
1
15
1
16
3
23
4
24
4
25
4
26
3
34
4
35
4
36
3
45
2
46
3
56
3
234
1
235
1
236
3
245
1
246
3
256
3
345
1
346
3
356
3
456
3
Legend:
x
y
x : jobs in the first region
y : denotes whether the node is evaluated or pruned
(designated by the condition number of the evaluation and pruning step)

Figure 22. Single-machine sequencing procedure enumeration tree of the example problem

6.3.2 Computational Experience

Although the number of sequences generated is cut down tremendously by the pruning step of the procedure, it still reaches quite a large value for problems with $N > 20$. The situation worsens if C' is in the neighborhood of $\frac{1}{2}\sum_{i=1}^{N} \mu_i$. To further investigate the performance of the procedure, the ratio of the best solution obtained by exploring the first 100 nodes of the procedure to the optimal solution was computed. In the experimentation, we assumed that $C' = C$, since $C \geq 20$ for all the problems. Three sets of problems with 10, 15 and 20 tasks were created, each set containing 10 problems. Due date was computed as $C = b \times [\sum_{i=1}^{N} \mu_i]$, and for each set three different values of b, namely, 0.25, 0.5 and 0.75 were used. Thus, a total of 90 problems were created and solved. In the test problems, $\mu_i \sim U[0;20]$ with $\sigma_i^2 = RAN_1(\mu_i)$ and $IC_i = RAN_2(\mu_i)$ where $RAN_1 \sim N[0.3;0.067]$ and $RAN_2 \sim N[0.05;0.01]$. The maximum, minimum and average ratio values for the problems solved are summarized in Table 27. If the ratio value is 1.00, then the solution obtained at the end of the 100th. node is either optimal or very close to the optimal value. As seen from the table, the procedure generates a solution that is within 0.2% of the optimal solution during the evaluation of the first 100 nodes. To put this in perspective, Table 28 depicts the maximum, minimum and average number of nodes evaluated in order to obtain the optimal solution for different problems. It should be noted that, to generate the first 100 nodes, it requires negligible computation time as compared to the enormous computation time required to obtain the optimal solution. Hence, the proposed procedure is very effective and generates almost optimal solutions in negligible amount of time.

Table 27. Ratios of the values of the solutions obtained at the end of the 100th node to that of the optimal solution

# of tasks	# of problems	$C = 0.25 \times \sum_{i=1}^{N} \mu_i$			# of tasks	# of problems	$C = 0.50 \times \sum_{i=1}^{N} \mu_i$			# of tasks	# of problems	$C = 0.75 \times \sum_{i=1}^{N} \mu_i$		
		Ratio					Ratio					Ratio		
		Ave.	Min.	Max.			Ave.	Min.	Max.			Ave.	Min.	Max.
10	10	1.0001	1.0000	1.0009	10	10	1.0000	1.0000	1.0002	10	10	1.0000	1.0000	1.0000
15	10	1.0002	1.0000	1.0004	15	10	1.0009	1.0000	1.0019	15	10	1.0029	1.0000	1.0136
20	10	1.0029	1.0003	1.0070	20	10	1.0022	1.0004	1.0091	20	10	1.0042	1.0022	1.0089

Table 28. Number of promising sequences generated to obtain the optimal solutions of the example problems

# of tasks	# of problems	$C = 0.25 \times \sum_{i=1}^{N} \mu_i$			# of tasks	# of problems	$C = 0.50 \times \sum_{i=1}^{N} \mu_i$			# of tasks	# of problems	$C = 0.75 \times \sum_{i=1}^{N} \mu_i$		
		Number of nodes					Number of nodes					Number of nodes		
		Ave.	Min.	Max.			Ave.	Min.	Max.			Ave.	Min.	Max.
10	10	59	30	126	10	10	123	54	166	10	10	64	24	88
15	10	950	634	1,272	15	10	3,056	842	4,976	15	10	650	296	1,248
20	10	10,540	7,296	17,059	20	10	53,499	15,122	102,714	20	10	9,864	5,733	18,244

6.3.3 Concluding Remarks On The Single-Machine Sequencing Procedure With Independent Tasks

For the problem of sequencing tasks on a single processor with a common due date and stochastic processing times, we have shown some conditions under which the sequence which minimizes the expected incompletion cost has tasks grouped in two regions. The first region has tasks ordered in the descending order of their incompletion costs while the second region has tasks ordered in the ascending order of their incompletion costs. The procedure developed to generate such sequences is quite efficient in the sense that it cuts down tremendously the number of sequences generated. For large problems, an approximate solution procedure has been shown to generate almost optimal sequences.

In the next section, we extend the procedure to construct a schedule on M parallel, identical machines

6.4 Extension Of The M-Machine Scheduling Procedure To Independent Tasks

The extension made to the single-machine sequencing procedure in the previous section is also applied to the M-machine scheduling procedure presented in Section 5.3.3.1. In other words, this version of the scheduling problem can be considered as the extension of the single-machine sequencing problem with independent tasks to M parallel machines.

In this section, we develop a procedure of constructing a schedule on M machines with independent tasks having a common due date and stochastic processing times. First, we describe the heuristic procedure, and the property which translates the single-machine sequence into a M-machine schedule is developed. Then, an analysis on the ratio of the heuristic and optimal solutions is presented. Finally, some computational experience on the performance of the procedure on some randomly generated problems is reported.

6.4.1 Development Of The M-Machine Scheduling Procedure With Independent Tasks

Consider an arbitrary schedule R in which tasks i and j are assigned to machines s and t, respectively, and $IC_i \geq IC_j$. Moreover, let's assume that $\mu_{Z_i} \leq \mu_{Z_j}$, where μ_{Z_i} is the sum of the expected processing times of the tasks preceding task i on the machine. In the schedule R′, the tasks i and j are interchanged in position. The situation is depicted in Figure 23. Let p_i and p_j be the incompletion probabilities of tasks i and j in schedule R, respectively, and let p'_i and p'_j denote the incompletion probabilities of tasks i and j in schedule R′.

The following Lemma determines the relation between the increase in the incompletion probability of task i and the decrease in the incompletion probability of task j as a result of interchanging their positions.

Lemma 6.1. If $\mu_{Z_i} + \mu_i \leq C'$, $\mu_{Z_j} + \mu_j \leq C'$, $\mu_i \geq \mu_j$ and $\mu_{Z_i} \leq \mu_{Z_j}$, then $p'_i - p_i \geq p_j - p'_j$. If $\mu_{Z_i} \geq C'$, $\mu_i \geq \mu_j$ and $\mu_{Z_i} \leq \mu_{Z_j}$, then $p'_i - p_i \leq p_j - p'_j$.

Proof. First note that the incompletion probability function for the Normal distribution is such that for $x \leq C'$, it is monotonically increasing and convex, and for $x > C'$, it is monotonically increasing and concave. The first case, namely, $\mu_{Z_i} + \mu_i \leq C'$ and $\mu_{Z_j} + \mu_j \leq C'$ of the Lemma

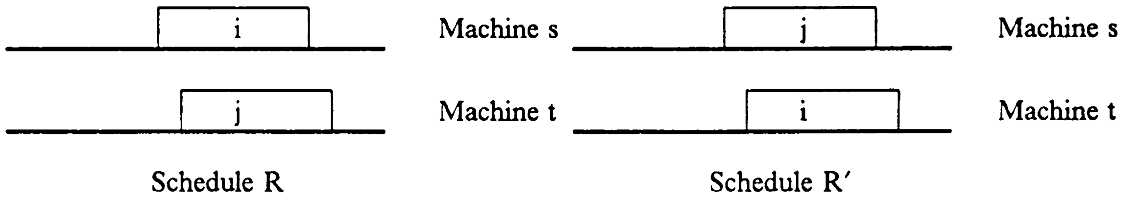


Figure 23. **Relative positions of tasks i and j in schedules R and R′**

belongs to the monotonically increasing and convex portion of the incompletion probability function and is shown in Figure 24. It depicts p_i, p'_i, p_j and p'_j . Since $\mu_i \geq \mu_j$ and $\mu_{Z_i} \leq \mu_{Z_j}$, $(p'_i - p_i)$ (designated by A) is clearly larger than $(p_j - p'_j)$ (designated by B).

The second case of the Lemma belongs to the monotonically increasing and concave portion of the incompletion probability function and consequently it follows that $p'_i - p_i \leq p_j - p'_j$.*

Next, we state a relationship between the contributions of the two tasks i and j to the total cost in schedules R and R′.

Theorem 6.3. If $\mu_{Z_i} + \mu_i \leq C'$, $\mu_{Z_j} + \mu_j \leq C'$, $\mu_{Z_i} \leq \mu_{Z_j}$ and $IC_i \geq IC_j$, then the contribution of the tasks i and j to the total cost in schedule R is less than or equal to that in schedule R′, and if $\mu_{Z_i} \geq C'$, $\mu_{Z_i} \leq \mu_{Z_j}$ and $IC_i \geq IC_j$, then the contribution in schedule R′ is less than or equal to that in schedule R.

Proof. Let Cost(R) be the contribution of the tasks i and j in schedule R. In the case where $\mu_{Z_i} + \mu_i \leq C'$, $\mu_{Z_j} + \mu_j \leq C'$ and $\mu_{Z_i} \leq \mu_{Z_j}$, it follows from Lemma 6.1 and $IC_i \geq IC_j$ that $IC_i\,(p'_i - p_i) \geq IC_j\,(p_j - p'_j)$ or $IC_i\,p'_i + IC_j\,p'_j \geq IC_i\,p_i + IC_j\,p_j$, thereby implying that $\mathrm{Cost}(R') \geq \mathrm{Cost}(R)$. Similarly, in the case where $\mu_{Z_i} \geq C'$, $\mu_{Z_i} \leq \mu_{Z_j}$ and $IC_i \geq IC_j$, it follows that $\mathrm{Cost}(R') \leq \mathrm{Cost}(R)$.*

In other words, the above result implies that for any pair of tasks i and j on any two machines, if $\mu_{Z_i} + \mu_i \leq C'$ and $\mu_{Z_j} + \mu_j \leq C'$, then the task with the larger incompletion cost should occupy the earlier position. If $\mu_{Z_i} \geq C'$ and $\mu_{Z_j} \geq C'$, then the task with the smaller incompletion cost should occupy the earlier position. Note that the above result is not valid for the cases when $\mu_{Z_i} + \mu_i \leq C'$ and $\mu_{Z_j} + \mu_j \geq C'$, or $\mu_{Z_i} + \mu_i \geq C'$ and $\mu_{Z_j} + \mu_j \leq C'$. The same is true for the cases when $\mu_{Z_i} \geq C'$ and $\mu_{Z_j} \leq C'$, or $\mu_{Z_i} \leq C'$ and $\mu_{Z_j} \geq C'$. Hence, this result determines the relative positions of the tasks in a M-machine schedule, although its optimality is not guaranteed.

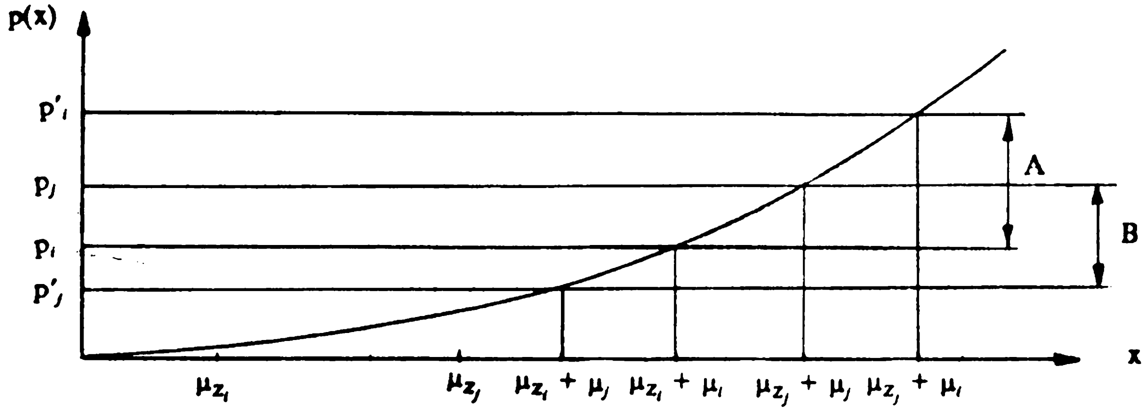


Figure 24. Incompletion probabilities of tasks i and j in schedules R and R′

The heuristic generates a M-machine schedule from the single machine sequence as follows:

Step 1. Set the due date equal to $M \times C'$. Obtain the single machine sequence for this due date with the single machine sequencing rule described in Section 6.3.

Step 2. Allocate tasks to M machines sequentially in their order of appearance in the single machine sequence by assigning the next task to the machine that has the least sum of the expected processing times of the tasks already assigned to it. Continue until all the tasks are assigned.

Step 3. The tasks within each machine are resequenced according to the single machine sequencing rule.

Step 2 of the heuristic follows from Theorem 6.3. Step 3 is applied to the schedule generated in Step 2, since the individual machine sequences obtained in Step 2 may not satisfy the single machine sequencing rule.

6.4.2 Analysis On The Ratio Of The Heuristic And Optimal Solutions

Consider the task i with an expected processing time of μ_i and partition it into h tasks. This is depicted in Figure 25. Note that the sum of the expected incompletion costs of the h tasks is equal to $\sum_{j=1}^{h} IC_j\, p_j$ and $p_h = p_i$.

The following Lemma determines the relation between the expected incompletion cost of task i and the sum of the expected incompletion costs of the h tasks.

Lemma 6.2. $IC_i\, p_i > \sum_{j=1}^{h} IC_j\, p_j \qquad$ for $h > 1$.

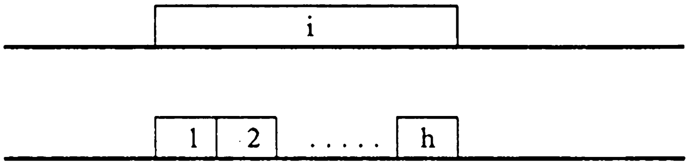


Figure 25. Partitioning of task i into h tasks

Proof. By construction, $\mu_i = \sum_{j=1}^{h} \mu_j$ and $p_i > p_j$ for $j = 1,\ldots,h-1$, and $p_i = p_j$ for $j = h$, since $\mu_{z_i} + \mu_i \geq \mu_{z_j} + \mu_j$ for $j = 1,\ldots,h-1$, and $\mu_{z_i} + \mu_i = \mu_{z_j} + \mu_j$ for $j = h$. Therefore, $\mu_i p_i > \sum_{j=1}^{h} \mu_j p_j$ for $h > 1$. Since $IC_i = k\,\mu_i$ for all i, it follows that $IC_i p_i > \sum_{j=1}^{h} IC_j p_j$ for $h > 1$.♦

Consider one of the h tasks and partition it into h′ tasks $(h' > 1)$. It follows from Lemma 6.2 that the sum of the expected incompletion costs of the h′ tasks is less than the expected incompletion cost of the original task. Thus, as the number of partitioned tasks increases, the expected incompletion cost of the sequence decreases. The following property summarizes the relation between the number of partitioning and the decrease in the expected incompletion cost of the sequence.

Property 6.1 In Lemma 6.2, as the number of partitionings of a task increases, the difference between $IC_i p_i$ and $\sum_{j=1}^{h} IC_j p_j$ also increases.

Let $\mu_{min} = \min_{j=1,\ldots,N} \{\mu_j\}$ and $\mu_{max} = \max_{j=1,\ldots,N} \{\mu_j\}$. Consider the machine j with r tasks on it. Let T_j denote the sum of the expected processing times of the tasks on machine j. Replace the tasks on machine j by $f(\frac{T_j}{\mu_{min}})$ tasks with expected processing times equal to μ_{min}, where f(x) denotes the largest integer less than or equal to x. Note that there will be $s = f(\frac{T_j}{\mu_{min}})$ tasks with expected processing times equal to μ_{min} and a fractional task with an expected processing time less than μ_{min} as the last task on machine j. Let this fractional task be denoted by k. The following Lemma determines the relation between the sum of the expected incompletion costs of the tasks on machine j and $\sum_{i=1}^{s} IC_i p_i + IC_k p_k$.

Lemma 6.3. $\sum_{i=1}^{s} IC_i p_i + IC_k p_k$ is a lower bound on the sum of the expected incompletion costs of the tasks on machine j.

Proof. Let the r tasks on machine j combined into a task with an expected processing time of T_j. Let this task be denoted by u. It follows from Lemma 6.2 that the expected incompletion cost of task u is an upper bound on the sum of the expected incompletion costs of the r tasks on machine j. Since $s \geq r$, it follows from Property 6.1 that the sum of the expected incompletion costs of the s tasks and task k is less than the sum of the expected incompletion costs of the r tasks on machine j. Therefore, it follows that $\sum_{i=1}^{s} IC_i p_i + IC_k p_k$ is a lower bound on the sum of the expected incompletion costs of the tasks on machine j.*

Consider a schedule that has $B_M = f(\frac{1}{M\mu_{min}} \sum_{i=1}^{N} \mu_i)$ tasks with expected processing times equal to μ_{min} on each machine. Let this schedule be denoted by $\Psi_{M,1}$. Similarly, let $\Psi_{M,2}$ denote the schedule that has $A_M = g(\frac{1}{M\mu_{max}} \sum_{i=1}^{N} \mu_i) + 1$ tasks with expected processing times equal to μ_{max} on each machine, where g(x) is the smallest integer larger than or equal to x. Note that $T_j = B_M \mu_{min}$ and $T_j = A_M \mu_{max}$ for all j in $\Psi_{M,1}$ and $\Psi_{M,2}$, respectively. Let $V(\Psi_{M,(.)})$ denote the cost of the schedule $\Psi_{M,(.)}$, and V_M denote the cost of any schedule on M machines of the N tasks; V_M^* denotes the cost of the optimal M-machine schedule. The following Corollary determines the relation between $V(\Psi_{M,1})$ and V_M .

Corollary 6.2. $V(\Psi_{M,1})$ constitutes a lower bound on V_M .

Proof. The proof of this Corollary is by construction as follows. Consider any schedule of the N tasks on M machines. Replace the tasks on machine j by $f(\frac{T_j}{\mu_{min}})$ tasks with expected processing times equal to μ_{min} , for j = 1,...,M. Thus, there will be $f(\frac{T_j}{\mu_{min}})$ tasks with expected processing times of μ_{min} and a task with an expected processing time less than μ_{min} may remain as the last task on machine j, for j = 1,...,M. It follows from Lemma 6.3 that the cost of this schedule is a lower bound on V_M . Let the fractional tasks on machines with large T_j 's be combined with the ones on machines with small T_j 's to form tasks with expected processing times equal to μ_{min} . This process obviously decreases the cost of the schedule. The cost of the schedule can be further decreased by transferring tasks from machines with larger number of tasks to machines with smaller number of tasks. If the number of tasks on each machine cannot be made equal to each other, then the last tasks of the machines with higher number of tasks are pruned, so that each machine would have the same number of tasks. Obviously, the cost of this schedule is a lower bound on V_M . On the other hand, the resulting schedule is equivalent to $\Psi_{M,1}$ and it follows that $V(\Psi_{M,1})$ is a lower bound on V_M .*

The following Corollary determines the relation between $V(\Psi_{M,2})$ and the cost of any schedule that satisfies Theorem 6.3.

Corollary 6.3. $V(\Psi_{M,2})$ constitutes an upper bound on the cost of any schedule of the N tasks that satisfies Theorem 6.3.

Proof. The proof of this Corollary also follows by construction. Consider any schedule of the problem that satisfies Theorem 6.3. Replace the tasks on machine j by $g(\frac{T_j}{\mu_{max}})$ tasks with expected processing times equal to μ_{max}, for $j = 1,\ldots,M$. It follows from Lemma 6.3 that the cost of this schedule is an upper bound on the cost of the original schedule. Note that the maximum difference between the number of tasks on the machines is one, since on a schedule that satisfies Theorem 6.3, $\max_{i,j=1,\ldots,N} \{T_i - T_j\} \leq \mu_{max}$. The number of tasks on each machine can be made equal to each other by appending a task with an expected processing time of μ_{max} to the machines with less number of tasks. Clearly, the cost of this schedule is also an upper bound on the cost of the original schedule. The resulting schedule has equal or less number of tasks on each machine than $\Psi_{M,2}$ Therefore, it follows that $V(\Psi_{M,2})$ is an upper bound on the cost of any schedule that satisfies Theorem 6.3.*

The incompletion probability function is depicted in Figure 26. A lower bound on the integral of p(x) from zero to D, for some $D > C'$, can be represented by the areas of the triangle KLC′ and trapezoid C′LPD as shown in Figure 26. In order to compute these areas, note that:

$$\text{Slope of the line KL} = Sl_{KL} = \frac{d}{dx}\left[\,1 - \Phi\left(\frac{C - x}{\sqrt{ax}}\right)\right]$$

where $x = C'$.

$$\frac{d}{dx}\left[\,1 - \Phi\left(\frac{C - x}{\sqrt{ax}}\right)\right] = \frac{e^{-\frac{(x-C)^2}{2ax}}}{\sqrt{2\pi}} \times \frac{(x + C)}{2\sqrt{a}\,x^{3/2}}$$

Substituting C′ instead of x yields the following expression for Sl_{KL} :

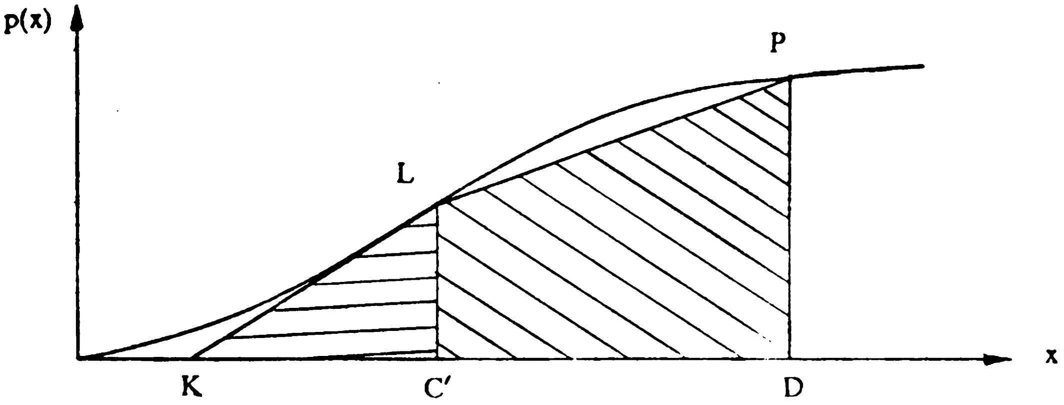


Figure 26. Incompletion probability function, p(x), and a lower bound on the area under p(x) from 0 to D > C′

$$\mathrm{Sl}_{KL} = \frac{e^{-\frac{(C'-C)^2}{2aC'}}}{\sqrt{2\pi}} \times \frac{(C' + C)}{2\sqrt{a}\ C'^{3/2}}$$

$$\text{Therefore, area of the triangle KLC}' = \frac{[\,p(C')\,]^2 \sqrt{2\pi a}\ C'^{3/2}}{(C' + C)\, e^{-\frac{(C'-C)^2}{2aC'}}}$$

$$\text{Area of the trapezoid C}'\text{LPD} = [D - C']\left[\frac{p(C')}{2} + \frac{1}{2}\Phi\left(\frac{D - C}{\sqrt{aD}}\right)\right]$$

Hence,

$$\int_0^D p(x).dx > \frac{[\,p(C')\,]^2 \sqrt{2\pi a}\ C'^{3/2}}{(C' + C)\, e^{-\frac{(C'-C)^2}{2aC'}}} + [D - C']\left[\frac{p(C')}{2} + \frac{1}{2}\Phi\left(\frac{D - C}{\sqrt{aD}}\right)\right] \quad \text{for } D > C' \qquad [6.14]$$

Let $\Psi_{M,0}$ be the schedule generated by the heuristic, then the ratio $\frac{V(\Psi_{M,0})}{V^*_M}$ is a measure of performance of the heuristic. Clearly, $\frac{V(\Psi_{M,0})}{V^*_M} \geq 1$. Theorem 6.4 states a worst-case upper bound on this ratio.

Theorem 6.4.

$$\frac{V(\Psi_{M,0})}{V^*_M} < \frac{2}{p(C') + 0.5}\left[1 + \frac{1.5\,\mu_{max} + C' + \mu_{min}}{\frac{1}{M}\sum_{i=1}^{N}\mu_i - C' - \mu_{min}}\right], \qquad \text{for } \frac{1}{M}\sum_{i=1}^{N}\mu_i > C' + \mu_{min}$$

Proof. From Corollaries 6.2 and 6.3, it follows that:

$$\frac{V(\Psi_{M,0})}{V^*_M} \leq \frac{V(\Psi_{M,2})}{V(\Psi_{M,1})} \qquad [6.15]$$

We will first define a lower bound on $V(\Psi_{M,1})$ and an upper bound on $V(\Psi_{M,2})$. $V(\Psi_{M,1})$ can be expressed as:

$$V(\Psi_{M,1}) = M\,k\,\mu_{min} \sum_{i=1}^{B_M} \Phi(\frac{i\,\mu_{min} - C}{\sqrt{i\,a\,\mu_{min}}})$$

Note that all tasks have an incompletion cost of $k\,\mu_{min}$. A lower bound on $V(\Psi_{M,1})$ can be expressed as follows:

$$V(\Psi_{M,1}) = M\,k\,\mu_{min} \sum_{i=1}^{B_M} \Phi(\frac{i\,\mu_{min} - C}{\sqrt{i\,a\,\mu_{min}}}) > M\,k \int_0^{U_M} p(x)\,dx \qquad [6.16]$$

where $U_M = \mu_{min}\,B_M$. The relation between $\mu_{min} \sum_{i=1}^{B_M} \Phi(\frac{i\,\mu_{min} - C}{\sqrt{i\,a\,\mu_{min}}})$ and the area under the incompletion probability function from zero to U_M is depicted in Figure 27. Following Expression [6.14], Expression [6.16] can be written as:

$$V(\Psi_{M,1}) > M\,k \left[\frac{[\,p(C')\,]^2 \sqrt{2\pi a}\; C'^{3/2}}{(C' + C)\,e^{-\frac{(C'-C)^2}{2aC'}}} + [U_M - C'] \left[\frac{p(C')}{2} + \frac{1}{2}\Phi(\frac{U_M - C}{\sqrt{aU_M}}) \right] \right], \; U_M > C'$$

Ignoring the first term and replacing $\Phi(\frac{U_M - C}{\sqrt{aU_M}})$ by its lower bound value of 0.5, we obtain:

$$V(\Psi_{M,1}) > M\,k \left[\frac{p(C')}{2} + 0.25 \right] [U_M - C'], \quad U_M > C' \qquad [6.17]$$

Since $U_M = \mu_{min}\, f(\frac{1}{M\mu_{min}} \sum_{i=1}^{N} \mu_i) \ge \mu_{min} (\frac{1}{M\mu_{min}} \sum_{i=1}^{N} \mu_i - 1)$, Expression [6.17] can be written as follows:

$$V(\Psi_{M,1}) > M\,k \left[\frac{p(C')}{2} + 0.25 \right] \left[\frac{1}{M} \sum_{i=1}^{N} \mu_i - C' - \mu_{min} \right], \text{ for } \frac{1}{M} \sum_{i=1}^{N} \mu_i > C' + \mu_{min} \qquad [6.18]$$

An upper bound on $V(\Psi_{M,2})$ can be derived using Corollary 6.3 as follows. Each machine in $\Psi_{M,2}$ has A_M tasks that have expected processing times equal to μ_{max} . Note that each task has an incompletion cost of $k\,\mu_{max}$. An upper bound on $V(\Psi_{M,2})$ can be written as:

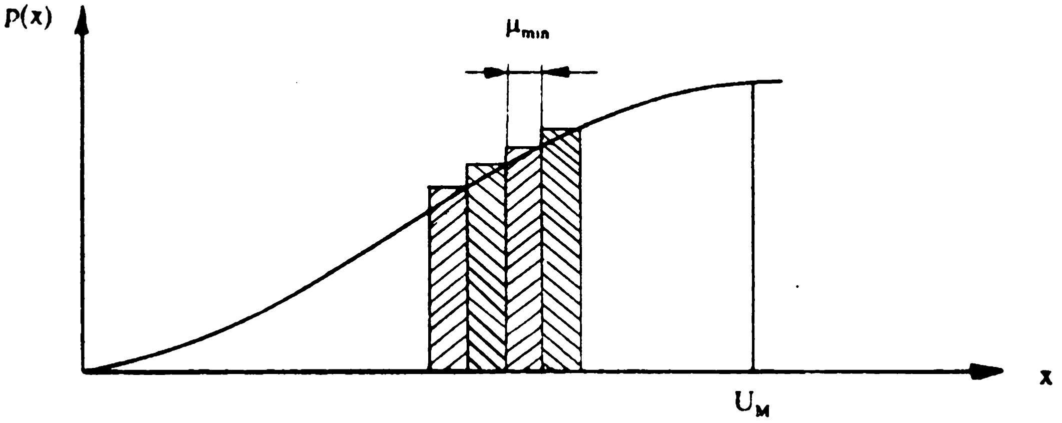


Figure 27. An upper bound on the area under the incompletion function, p(x)

$$V(\Psi_{M,2}) \leq M\,k\,\mu_{max} \sum_{i=1}^{A_M} \Phi\left(\frac{i\,\mu_{max} - C}{\sqrt{i\,a\,\mu_{max}}}\right) \qquad [6.19]$$

Since $\Phi\left(\frac{i\,\mu_{max} - C}{\sqrt{i\,a\,\mu_{max}}}\right) \leq 0.5$ for $i = 1$, and $\Phi\left(\frac{i\,\mu_{max} - C}{\sqrt{i\,a\,\mu_{max}}}\right) < 1.0$ for $i = 2,..,A_M$, we can write the following expression:

$$\sum_{i=1}^{A_M} \Phi\left(\frac{i\,\mu_{max} - C}{\sqrt{i\,a\,\mu_{max}}}\right) < 0.5 + (A_M - 1) = A_M - 0.5 \qquad [6.20]$$

Substituting Expression [6.20] into Expression [6.19] yields:

$$V(\Psi_{M,2}) < M\,k\,\mu_{max}\,(A_M - 0.5) \qquad [6.21]$$

Since $A_M = g\left(\frac{1}{M\mu_{max}} \sum_{i=1}^{N} \mu_i\right) + 1 \leq \frac{1}{M\mu_{max}} \sum_{i=1}^{N} \mu_i + 2$, Expression [6.21] can be written as follows:

$$V(\Psi_{M,2}) < M\,k\,\mu_{max}\,(A_M - 1.5) \leq M\,k\,\mu_{max}\left[\frac{1}{M\mu_{max}} \sum_{i=1}^{N} \mu_i + 1.5\right] \qquad [6.22]$$

By substituting the relationships given by Expressions [6.18] and [6.22] into Expression [6.15], we obtain,

$$\frac{V(\Psi_{M,0})}{V_M^*} \leq \frac{V(\Psi_{M,2})}{V(\Psi_{M,1})} < \frac{M\,k\,\mu_{max}\left[\frac{1}{M\mu_{max}} \sum_{i=1}^{N} \mu_i + 1.5\right]}{M\,k\left[\frac{p(C')}{2} + 0.25\right]\left[\frac{1}{M} \sum_{i=1}^{N} \mu_i - C' - \mu_{min}\right]} =$$

$$= \frac{2}{p(C') + 0.5}\left[1 + \frac{1.5\,\mu_{max} + C' + \mu_{min}}{\frac{1}{M} \sum_{i=1}^{N} \mu_i - C' - \mu_{min}}\right]$$

Note that the bound derived in Theorem 6.4 is defined for $\frac{1}{M} \sum_{i=1}^{N} \mu_i > C' + \mu_{min}$. When $\frac{1}{M} \sum_{i=1}^{N} \mu_i \leq C' + \mu_{min}$, the problem is trivial to solve as shown in Section 6.3, because the optimal solution can be obtained simply by assigning the larger tasks to the earliest available position on any machine. Although the value of the bound increases as $\frac{1}{M} \sum_{i=1}^{N} \mu_i$ approaches $C' + \mu_{min}$, it remains at quite acceptable values for practical problem parameters. Also note that the value of the bound is an upper bound on the worst-case performance of the heuristic, since it is determined considering the two extreme cases simultaneously, namely, pertaining to the schedules $\Psi_{M,1}$ and $\Psi_{M,2}$. As can be easily seen, the value of the bound increases with increases in C′, M, μ_{min} and μ_{max} , but it decreases with an increase in $\sum_{i=1}^{N} \mu_i$. In other words, as the problem size increases, the value of the bound decreases. This is an attractive feature of the bound and is summarized in the following Corollary.

Corollary 6.4. The value of the bound derived in Theorem 6.4 decreases as the problem size increases.

For sufficiently large C and small a values, it is shown in Section 6.3.1 that C′ and C are quite close to each other. Assuming C′ = C and since p(x = C) = 0.5, then the value of the bound derived in Theorem 6.4 reduces to the following expression:

$$\frac{V(\Psi_{M,0})}{V_M^*} < 2\left[1 + \frac{1.5\,\mu_{max} + C + \mu_{min}}{\frac{1}{M}\sum_{i=1}^{N} \mu_i - C - \mu_{min}}\right]$$

6.4.3 Performance Of The M-Machine Scheduling Procedure With Independent Tasks

The analysis presented in the previous section is a worst-case analysis on the ratio of the heuristic and optimal solutions as it was based on two extreme cases pertaining to $\Psi_{M,1}$ and $\Psi_{M,2}$. For a specific problem, a bound on the performance of the heuristic can be derived as follows.

$$\frac{V(\Psi_{M,0})}{V_M^*} \leq \frac{V(\Psi_{M,0})}{V(\Psi_{M,1})}$$

The bound given above is valid since $V(\Psi_{M,1})$ constitutes a lower bound on V_M^* , as shown in Corollary 6.2.

To investigate the performance of the heuristic, the bound above was computed for several randomly generated problems. In the experimentation, we assumed that $C' = C$, since the C values of all the problems were sufficiently large. Three sets of problems with 20, 30 and 40 tasks were created. 20-task problems were solved for 2, 3 and 4 machines, 30-task problems were solved for 2, 3, 4, 5 and 6 machines and 40-task problems were solved for 2, 3, 4, 5, 6 and 7 machines. For each number of machines, 10 problems were created. Due date was computed as $C = b \times [\sum_{i=1}^{N} \mu_i]$, and for each set three different values of b, namely, 0.25, 0.5 and 0.75 were used. Thus, a total of 420 problems were created and solved. In the test problems, $\mu_i \sim U[0;20]$ with $\sigma_i^2 = RAN_1(\mu_i)$ and $IC_i = RAN_2(\mu_i)$ where $RAN_1 \sim N[0.3;0.067]$ and $RAN_2 \sim N[0.05;0.01]$. The bounds on the ratio of the values of the heuristic solution and the optimal solution for the problems solved are summarized in Table 29. For each set of problems, the average, minimum and maximum bound values are given. As it is seen from Table 29, the heuristic procedure generates almost optimal solutions. Moreover, computationally, it is very easy to use.

6.4.4 Concluding Remarks On The M-Machine Scheduling Procedure With Independent Tasks

In this section we presented a heuristic procedure for the problem of scheduling N tasks on M machines with the objective of minimizing the expected incompletion cost for the case in which incompletion costs of the tasks are proportional to their expected performance times. The heuristic procedure constructs the M machine schedule from a single machine sequence of the N tasks. Computational experience indicates that the heuristic procedure generates almost optimal solutions and it is very easy to use. An analysis of the worst-case value of the ratio of the heuristic and op-

Table 29. Ratios of the heuristic solution value to that of the optimal solution value

			$C = 0.25 \times \sum_{i=1}^{N} \mu_i$						$C = 0.50 \times \sum_{i=1}^{N} \mu_i$						$C = 0.75 \times \sum_{i=1}^{N} \mu_i$		
# of tasks	# of mach	# of prob.	Value of the ratio			# of tasks	# of mach	# of prob.	Value of the ratio			# of tasks	# of mach	# of prob.	Value of the ratio		
			Ave.	Min.	Max.				Ave.	Min.	Max.				Ave.	Min.	Max.
20	2	10	1.025	1.015	1.043	20	2	10	1.044	1.014	1.055	20	2	10	1.094	1.066	1.123
	3	10	1.049	1.036	1.072		3	10	1.075	1.044	1.106		3	10	1.172	1.118	1.230
	4	10	1.086	1.054	1.141		4	10	1.117	1.054	1.178		4	10	1.236	1.177	1.302
30	2	10	1.014	1.011	1.016	30	2	10	1.028	1.023	1.033	30	2	10	1.055	1.034	1.062
	3	10	1.026	1.021	1.033		3	10	1.045	1.030	1.056		3	10	1.097	1.054	1.121
	4	10	1.032	1.027	1.036		4	10	1.066	1.053	1.079		4	10	1.134	1.065	1.171
	5	10	1.061	1.045	1.080		5	10	1.091	1.072	1.104		5	10	1.185	1.147	1.210
	6	10	1.077	1.066	1.115		6	10	1.117	1.080	1.145		6	10	1.245	1.149	1.313
40	2	10	1.013	1.009	1.018	40	2	10	1.020	1.014	1.029	40	2	10	1.039	1.028	1.054
	3	10	1.020	1.018	1.024		3	10	1.030	1.026	1.035		3	10	1.062	1.042	1.081
	4	10	1.026	1.019	1.030		4	10	1.042	1.034	1.049		4	10	1.087	1.054	1.111
	5	10	1.032	1.024	1.043		5	10	1.059	1.044	1.076		5	10	1.111	1.087	1.146
	6	10	1.047	1.036	1.063		6	10	1.065	1.044	1.091		6	10	1.153	1.096	1.201
	7	10	1.067	1.053	1.082		7	10	1.094	1.075	1.116		7	10	1.186	1.162	1.226

timal solutions is presented and the worst-case value is shown to be finite. Moreover, the value of the ratio decreases as the problem size increases.

7.0 Summary, Conclusions And Recommendations

7.1 *Summary*

A procedure to solve the stochastic, single-model assembly line balancing problem is developed and presented in this dissertation. Its basic characteristic is that the problem is divided into subproblems and the solutions of the subproblems are combined to form the solution of the original problem.

First, a dynamic programming formulation of the problem is developed. The formulation is effective only for problems of limited sizes. Thus, a bounding strategy is incorporated with the formulation to enable the procedure to solve problems of larger sizes.

An approximation procedure is developed that divides the assembly line balancing problem into subproblems and applies the improvement procedure to each subproblem. The approximate solutions of the subproblems obtained using the dynamic programming procedure constitute the initial solutions to the improvement procedure. This procedure either improves the initial solutions or determines that they are close to the optimal ones. A detailed experimentation is carried out to investigate several aspects of the approximation procedure. The improvement procedure solutions are then combined to form the solution of the original problem.

The improvement procedure utilizes specific sequencing and scheduling rules. A single-machine sequencing procedure is developed for the objective of minimizing the expected incompletion cost. This procedure is then extended to construct a schedule on M parallel machines. Heuristic solution procedures are also developed for the sequencing and scheduling problems for the special cases in which the incompletion costs of the tasks are proportional to their expected performance times. Computational results and analyses indicate that these procedures result in almost optimal solutions.

7.2 Conclusions

Surveys on the manufacturing industry indicate that, in general, the currently available techniques to balance assembly lines are not used. It suggests that either the currently available techniques are inadequate to model the actual conditions of assembly lines or the practitioners are unfamiliar with the published algorithms. The methodology developed in this dissertation could be considered as an approach to model the conditions of assembly lines and obtain a solution close to the optimal one. Although the problem has been addressed before in the literature, a cost model that captures all the interactions of the tasks allocated to stations was nonexistent. A new cost model is developed to compute the total system cost of a line for a fixed number of stations and allocations of tasks to stations. Solution procedures developed for the problem utilize new and different heuristic rules, and the experimentations carried out indicate that the results are better than those of the other proedures reported in the literature. New heuristic procedures are also developed for the problem obtained by relaxing the precedence constraints among the tasks, and the analyses indicate that these procedures result in almost optimal solutions.

The following conclusions are drawn from the analyses and experimentations made on the procedures developed:

1. The research aim was to develop and implement a methodology to solve the stochastic, single-model assembly line balancing problem for the objective of minimizing the total system cost. An effective procedure is developed for problems of realistic sizes. Although the solution obtained is not guaranteed to be the optimal one, it lies within a small neighborhood of the optimal solution.

2. A new cost model of the problem for a given number of stations and allocations of tasks to stations is developed. The interactions of the tasks assigned to the same station and the interactions of the tasks assigned to different stations complicate the structure of the problem considerably. A special probability enumeration tree is utilized to determine several variables necessary to compute the total system cost of a line.

3. Dynamic programming formulation of the problem obtains the optimal solution if carried to completion. For problems of large sizes, the procedure requires excessive storage and computations. Therefore, a bounding strategy is incorporated with the procedure so that a solution is obtained for problems of larger sizes. On the other hand, the solution obtained with the bounding strategy is no longer the optimal one. The experimentation reveals that the bounding strategy is very effective and the solutions obtained are quite close to the optimal ones. Since the procedure generates good solutions, they constitute good initial solutions to the improvement procedure. A comparison of the dynamic programming procedure with the bounding strategy and the technique of Kottas and Lau indicates that the dynamic programming procedure results in as good solutions as the technique of Kottas and Lau.

4. An improvement procedure is developed that improves the approximate solution obtained using the dynamic programming procedure with the bounding strategy or determines that it is

close to the optimal one. The experimental results show that the procedure is quite effective in improving the initial solutions for problems with 20 or less number tasks.

5. The improvement procedure results in solutions better than the technique of Kottas and Lau and the dynamic programming procedure with the bounding strategy. The solutions of the dynamic programming procedure with the bounding strategy lie between the solutions of the other two procedures. The difference between the solutions of the procedures decreases as the incompletion costs of the tasks get large relative to the labor rate. The technique of Kottas and Lau and the dynamic programming procedure with the bounding strategy result in solutions relatively better for higher magnitudes of the incompletion costs, whereas the improvement procedure results in ε-optimal solutions for all magnitudes of the incompletion costs.

6. For problems with 20 or more number of tasks, an approximation procedure is developed that divides the problem into subproblems with 20 or less number of tasks. The experimental results indicate that the procedure generates as good or better solutions as the technique of Kottas and Lau. The technique of Kottas and Lau is the only reported technique to solve the problem for the objective of minimizing the total expected operating cost.

7. A heuristic procedure is developed for the single-machine sequencing problem with N tasks which have a common due date and stochastic processing times for the objective of minimizing the expected incompletion cost. An approximate solution procedure is developed for the rule; the experimentation carried out indicates that the procedure generates almost optimal solutions and is computationally attractive.

8. A heuristic procedure is also developed that constructs a schedule for N tasks with stochastic processing times and a common due date on M parallel, identical machines for the objective of minimizing the total expected incompletion cost. A worst-case analysis on the ratio of the heuristic and optimal solutions is made and a bound on the ratio is derived. The bound is

shown to be finite and its value decreases as the problem size increases. The experimental results indicate that the heuristic procedure generates almost optimal solutions.

7.3 Recommendations for further research

Research into stochastic assembly line balancing problem is far from complete. Different versions of the problem require more in depth analysis and research than presented herein. This research has concentrated on developing an effective methodology for the single-model, stochastic version of the problem. The results indicate that realistically complex, large-scale problems can be solved within the existing computer facilities. Since effective solution procedures for the other versions of the problem are nonexistent, it should be taken as evidence for initiating more streneous work in this area. In addition, extensions of the problem find applications in several areas, especially in sequencing and scheduling tasks among machines. More research is required to generalize these extensions.

Some research areas for further investigation related to the study performed are itemized below:

1. The single-model, stochastic version of the problem is the most simple version to analyze. On the other hand, industry utilizes mixed and multi-model lines quite frequently. Thus, effective methodologies are needed for the mixed and multi-model versions of the stochastic assembly line balancing problem.

2. The objective function of the model could be modified to include other cost factors. Some tasks could be performed in several ways using different types of equipment. The costs associated with the type of equipment for such tasks could be considered in the objective function.

Note that the decision of choosing the type of equipment for a task should not be made separately, since the selection affects the performance time of the task.

3. More research is needed to develop methodologies for the stochastic assembly line balancing problem in which in-process inventories are allowed. This version of the problem is also frequently used in industry and an effective solution procedure is nonexistent.

4. A relationship between the cycle time and the total system cost should be developed. When the assumption that the demand rate is known with certainty is relaxed, the cycle time no longer remains to be a constant. If such a relationship is obtained, it becomes possible to determine the cycle time that minimizes the total system cost.

5. The sequencing and scheduling procedures developed for the extensions of the problem should be analyzed for the case in which each task has its own due date. Thus, such a generalization would enlarge the application areas of these procedures significantly.

Bibliography

1. Agrawal, P.K.,"The related activity concept in assembly line balancing", *International Journal of Production Research*, Vol.23, No.2, 1985, pp.403-421.

2. Akagi, F., Osaki, H., and S. Kikuchi, "A method for assembly line balancing with more than one worker in each station", *International Journal of Production Research*, Vol.21, No.5, 1983, pp.755-770.

3. Arcus, A.L., "COMSOAL: A computer method of sequencing operations for assembly lines", *International Journal of Production Research*, Vol.4, No.4, 1966, pp.259-277.

4. Assche, F.V., and W.S. Herroelen, "An optimal procedure for the single-model deterministic assembly line balancing problem", *European Journal of Operational Research*, Vol.3, No.2, 1979, pp.142-149.

5. Bagga, P.C., and K.R. Kalra, "A node elimination procedure for Townsend's algorithm for solving the single machine quadratic penalty function scheduling problem", *Management Science*, Vol.26, No.6, 1980, pp.633-636.

6. Baker, K.R., and L.E. Schrage, "Finding an optimal sequence by dynamic programming: An extension to precedence related tasks", *Operations Research*, Vol.26, No.1, 1978, pp.111-120.

7. Banarjee, B.P., "Single facility sequencing with random execution times", *Operations Research*, Vol.13, No.3., 1965, pp.358-364.

8. Bennett, G.B., and J. Byrd, "A trainable heuristic procedure for the assembly line balancing problem", *AIIE Transactions*, Vol.8, No.2, 1976, pp.195-201.

9. Bowman, E.H., "Assembly-line balancing by linear programming", *Operations Research*, Vol.8, No.3, 1960, pp.385-389.

10. Bruno, J.L. and P.J. Downey, "Probabilistic bounds on the performance of list scheduling", *SIAM Journal on Computing*, Vol.15, No.2, 1986, pp.409-417.

11. Buxey, G.M., Slack, N.D., and R. Wild, "Production flow line system design - a review", *AIIE Transactions*, Vol.5, No.1, 1973, pp.37-48.

12. Buxey, G.M., "Assembly line balancing with multiple stations", *Management Science*, Vol.20, No.6, 1974, pp.1010-1021.

13. Buxey, G., "Incompletion costs versus labour efficiency on the fixed-item moving belt flowline", *International Journal of Production Research*, Vol.16, No.3, 1978, pp.233-247.

14. Caruso, F.R., "Assembly line balancing for improved profits", *Automation*, 1965, pp.48-52.

15. Chakravarty, A.K. and A. Shtub, "Balancing mixed model lines with in-process inventories", *Management Science*, Vol.31, No.9, 1985, pp.1161-1174.

16. Chakravarty, A.K. and A. Shtub, "A cost minimization procedure for mixed model production lines with normally distributed task times" *European Journal of Operational Research*, Vol.23, No.1, 1986, pp.25-36.

17. Chase, R.B., "Survey of paced assembly lines", *Industrial Engineering*, Vol.6, 1974, pp.14-18.

18. Coffman, E.G. and E.N. Gilbert, "On the expected relative performance of list scheduling", *Operations Research*, Vol.33, No.3, 1985, pp.548-561.

19. Cooper, L., and M.W. Cooper, "Introduction to dynamic programming", Pergamon Press, Ltd., New York, 1981.

20. Dar-El, E.M., "MALB - a heuristic technique for balancing large single-model assembly lines", *AIIE Transactions*, Vol.5, No.4, 1973, pp.343-356.

21. Dar-El, E.M., "Solving large single-model assembly line balancing problems - a comparative study", *AIIE Transactions*, Vol.7, No.3, 1975, pp.302-310.

22. Dar-El, E.M., "Mixed-model assembly line sequencing problems", *OMEGA*, Vol.6, No.4, 1978, pp.313-323.

23. Dar-El, E.M., and R.F. Cother, "Assembly line sequencing for model mix", *International Journal of Production Research*, Vol.13, No.5, 1975, pp.463-477.

24. Dar-El, E.M., and S. Cucuy, "Optimal mixed-model sequencing for balanced assembly lines", *Omega*, Vol.5, No.3, 1977, pp.333-342.

25. Dar-El, E.M., and Y. Rubinovitch, "MUST - a multiple solutions technique for balancing single model assembly lines", *Management Science*, Vol.25, No.11, 1979, pp.1105-1114.

26. Davis, K.R., and L.F. Simmons, "Improving assembly line efficiency: A dynamic programming-heuristic approach", *Computers and Operations Research*, Vol.4, No.2, 1977, pp.75-87.

27. Dogramaci, A. and J. Surkis, "Evaluation of a heuristic for scheduling independent jobs on parallel identical processors", *Management Science*, Vol.25 No.12, 1979, pp.1208-1216.

28. Driscoll, J. and A.A.A. Abdel-Shafi, "A simulation approach to evaluating assembly line balancing solutions", *International Journal of Production Research*, Vol.23, No.5, 1985, pp.975-985.

29. Eastman, W.L., Even, S. and I.M. Isaacs, "Bounds for the optimal scheduling of n jobs on m processors", *Management Science*, Vol.11 No.2, 1964, pp.268-279.

30. Elmaghraby, S.E. and S.H. Park, "Scheduling jobs on a number of identical machines", *AIIE Transactions*, Vol.6, No.1, 1974, pp.1-13.

31. El-Rayah, T.E., "The efficiency of balanced and unbalanced production lines", *International Journal of Production Research*, Vol.17, No.1, 1979, pp.61-75.

32. Freeman, D.R., and J.V. Jucker, "The line balancing problem", *The Journal of Industrial Engineering*, Vol.18, No.6, 1967, pp.361-364.

33. Garey, M.R. and R.L. Graham, "Bounds for multiprocessor scheduling with resource constraints", *SIAM Journal on Computing*, Vol.4, No.2, 1975, pp.187-200.

34. Gehrlein, W.V. and J.H. Patterson, "Balancing single-model assembly lines: Comments on a paper by E.M. Dar-El (Mansoor)", *AIIE Transactions*, Vol.10, No.1, 1978, pp.109-112.

35. Graham, R.L., "Bounds on multiprocessing timing anomalies", *SIAM Journal on Applied Mathematics*, Vol.17, No.2, 1969, pp.416-429.

36. Gupta, J.N.D. and A.R. Maykut, "Scheduling jobs on parallel processors with dynamic programming", *Decision Sciences*, Vol.4, No.4, 1973, pp.447-457.

37.Gupta, S.K., and Sen, T., "On the single-machine scheduling problem with quadratic penalty function of completion times: An improved branching procedure", *Management Science* Vol.30, No.5, 1984, pp.644-647.

38. Gutjahr, A.L., and G.L. Nemhauser, "An algorithm for the line balancing problem", *Management Science*, Vol.11, No.2, 1964, pp.308-315.

39. Held, M., Karp, R.M., and R. Shareshian,"Assembly-line balancing - Dynamic programming with precedence constraints", *Operations Research*, Vol.11, No.3, 1963, pp.442-459.

40. Helgeson, W.B., and D.P. Birnie,"Assembly line balancing using the ranked positional weight technique", *The Journal of Industrial Engineering*, Vol.12, No.6, 1961, pp.394-398.

41. Hillier, F.S., and G.J. Lieberman, "Introduction to Operations Research", Holden-Day, Inc., Third edition, San Fransisco, 1980.

42. Hoffmann, T.R., "Assembly line balancing with a precedence matrix", *Management Science*, Vol.9, No.4, 1963, pp.551-562.

43. Hu, T.C., "Parallel sequencing and assembly line problems", *Operations Research*, Vol.9, No.6, 1961, pp.841-848.

44. Ignall, E.J., "A review of assembly line balancing", *The Journal of Industrial Engineering*, Vol.16, No.4, 1965, pp.244-254.

45. Jackson, J.R., "A computing procedure for a line balancing problem", *Management Science*, Vol.2, No.3, 1956, pp.261-271.

46. Johnson, R.V., "Assembly line balancing algorithms: computation comparisons", *International Journal of Production Research*, Vol.19, No.3, 1981, pp.277-287.

47. Johnson, R.V., "A branch and bound algorithm for assembly line balancing problems with formulation irregularities", *Management Science*, Vol.29, No.11, 1983, pp.1309-1324.

48. Johnson, L.A. and D.C. Montgomery, "Operations research in production planning, scheduling, and inventory control", John Wiley & Sons, Inc., New York, 1974.

49. Kao, E.P.C., "A preference order dynamic program for stochastic assembly line balancing", *Management Science*, Vol.22, No.10, 1976, pp.1097-1104.

50. Kao, E.P.C., "Computational experience with a stochastic assembly line balancing algorithm", *Computers and Operations Research*, Vol.6, No.2, 1979, pp.79-86.

51. Kao, E.P.C., and M. Queyranne, "On dynamic programming methods for assembly line balancing", *Operations Research*, Vol.30, No.1, 1982, pp.375-390.

52. Kilbridge, M.D., and L. Wester, "The balance delay problem", *Management Science*, Vol.8, No.1, 1961, pp.69-84.

53. Kilbridge, M.D., and L. Wester, "A heuristic method of assembly line balancing", *The Journal of Industrial Engineering*, Vol.12, No.4, 1961, pp.292-298.

54. Kilbridge, M.D., and L. Wester, "Heuristic line balancing: A case", *The Journal of Industrial Engineering*, Vol.13, No.2, 1962, pp.139-149.

55. Kilbridge, M.D., and L. Wester, "A review of analytical systems of line balancing", *Operations Research*, Vol.10, No.5, 1962, pp.626-638.

56. Klein, M., "On assembly line balancing", *Operations Research*, Vol.11, No.2, 1963, pp.274-281.

57. Kottas, J.F., and H.S. Lau,"A cost oriented approach to stochastic line balancing", *AIIE Transactions*, Vol.5, No.2, 1973, pp.164-171.

58. Kottas, J.F., and H.S. Lau, "A total operating cost model for paced lines with stochastic task times", *AIIE Transactions*, Vol.8, No.2, 1976, pp.234-240.

59. Kottas, J.F., and H.S. Lau, "A stochastic line balancing procedure", *International Journal of Production Research*, Vol.19, No.2, 1981, pp.177-193.

60. Lawler, E.L., "On scheduling problems with deferral costs", *Management Science*, Vol.11, 1964, pp.280-288.

61. Lehman, M., "On criteria for assigning models to assembly lines", *International Journal of Production Research*, Vol.7, No.4, 1969, pp.269-285.

62. Loulou, R., "Tight bounds and probabilistic analysis of two heuristics for parallel processor scheduling", *Mathematics of Operations Research*, Vol.9, No.1, 1984, pp.142-150.

63. Macaskill, J.L.C., "Production-line balances for mixed-model lines", *Management Science*, Vol.19, No.4, 1972, pp.423-434.

64. Mansoor, E.M., and S.Ben-Tuvia, "Optimizing balanced assembly lines", *The Journal of Industrial Engineering*, Vol.17, No.3, 1966, pp.126-131.

65. Mastor, A.A., "An experimental investigation and comparative evaluation of production line balancing techniques", *Management Science*, Vol.16, No.11, 1970, pp.728-746.

66. McNaughton, R., "Scheduling with deadlines and loss functions", *Management Science*, Vol.6, 1959, pp.1-12.

67. Moberly, L.E., and F.P. Wyman, "An application of simulation to the comparison of assembly line configurations", *Decision Sciences*, Vol.4, No.4, 1973, pp.505-516.

68. Moodie, C.L., and H.H. Young, "A heuristic method of assembly line balancing for assumptions of constant or variable work element times", *The Journal of Industrial Engineering*, Vol.16, No.1, 1965, pp.23-29.

69. Nevins, A.J., "Assembly line balancing using best bud search", *Management Science*, Vol.18, No.9, 1972, pp.529-539.

70. Patterson J.H., and J.J. Albracht, "Assembly-line balancing: Zero-one programming with Fibonacci search", *Operations Research*, Vol.23, No.1, 1975, pp.166-172.

71. Pinedo, M., "Stochastic scheduling with release dates and due dates", *Operations Research*, Vol.31, No.3, 1983, pp.559-572.

72. Pinedo, M., and G. Weiss, "Scheduling jobs with exponentially distributed processing times on two machines with resource constraints", *Management Science*, Vol.30, No.7, 1984, pp.883-889.

73. Pinto, P.A., Dannenbring, D.G., and B.M. Khumawala, "A branch and bound algorithm for assembly line balancing with paralleling", *International Journal of Production Research*, Vol.13, No.2, 1975, pp.183-196.

74. Pinto, P.A., Dannenbring, D.G., and B.M. Khumawala, "A heuristic network procedure for the assembly line balancing problem", *Naval Research Logistics Review*, Vol.25, No.2, 1978, pp.299-307.

75. Pinto, P.A., Dannenbring, D.G., and B.M. Khumawala, "Branch and Bound and heuristic procedures for assembly line balancing with paralleling of stations", *International Journal of Production Research*, Vol.19, No.5, 1981, pp.565-576.

76. Pinto, P.A., Dannenbring, D.G., and B.M. Khumawala, "Assembly line balancing with processing alternatives: An application", *Management Science*, Vol.29, No.7, 1983, pp.817-830.

77. Raouf, A., El-Sayed, E.A. and C.L. Tsui, "A new heuristic approach to assembly line balancing", *Computers and Industrial Engineering*, Vol.4, No.3, 1980, pp.223-234.

78. Raouf, A. and C.L. Tsui, "A new method for assembly line balancing having stochastic work elements", *Computers and Industrial Engineering*, Vol.6, No.2, 1982, pp.131-148.

79. Reeve, N.R., and W.H. Thomas,"Balancing stochastic assembly lines", *AIIE Transactions*, Vol.5, No.3, 1973, pp.223-229.

80. Reiter, R., "On assembly-line balancing problems", *Operations Research*, Vol.17, No.4, 1969, pp.685-700.

81. Roberts, S.D., and C.D. Villa,"On a multiproduct assembly line-balancing problem", *AIIE Transactions*, Vol.2, No.4, 1970, pp.361-364.

82. Rosenblatt, M.J., and R.C. Carlson, "Designing a production line to maximize profit", *IIE Transactions*, Vol.17, No.2, 1985, pp.117-121.

83. Root, J.G., "Scheduling with deadlines and loss functions on k parallel machines", *Management Science*, Vol.11, No.3, 1965, pp.460-475.

84. Rothkoph, M.H., "Scheduling independent tasks on parallel processors", *Management Science*, Vol.12, No.5, 1966, pp.437-447.

85. Salveson, M.E.,"The assembly line balancing problem", *The Journal of Industrial Engineering*, Vol.6, No.3, 1955, pp.18-25.

86. Sarin, S.C. and S.E. Elmaghraby, "Bounds on the performance of a heuristic to schedule precedence-related jobs on parallel machines", *International Journal of Production Research*, Vol.22, No.1, 1984, pp.17-30.

87. Sarker, B.R., and J.G. Shanthikumar, "A generalized approach for serial or parallel line balancing", *International Journal of Production Research*, Vol.21, No.1, 1983, pp.109-133.

88. Sawyer, J.H.F., "Line balancing", Machinery Publishing Co.Ltd., Great Britain, 1970.

89. Schild, A., and I.J. Fredman, "Scheduling tasks with deadlines and non-linear loss functions", *Management Science*, Vol.9, 1962, pp.73-81.

90. Schrage, L. and K.R. Baker, "Dynamic programming solution of sequencing problems with precedence constraints", *Operations Research*, Vol.26, No.3, 1978, pp.444-449.

91. Sculli, D., "Dynamic aspects of line balancing", *OMEGA*, Vol.7, No.6, 1979, pp.557-561.

92. Sculli, D., "Short term adjustments to production lines", *Computers and Industrial Engineering* Vol.8, No.1, 1984, pp.53--63.

93. Shtub, A., "The effect of incompletion cost on the line balancing with multiple manning of work stations", *International Journal of Production Research*, Vol.22, No.2, 1984, pp.235-245.

94. Sphicas, G.P., and F.N. Silverman, "Deterministic equivalents for stochastic assembly line balancing", *AIIE Transactions*, Vol.8, No.2, 1976, pp.280-282.

95. Steiner, G., "Single machine scheduling with precedence constraints of dimension two", *Mathematics of Operations Research*, Vol.9, No.2, 1984, pp.248-259.

96. Sury, R.J., "Aspects of assembly line balancing", *International Journal of Production Research*, Vol.9, No.4, 1971, pp.501-512.

97. Talbot, F.B., and J.H. Patterson, "An integer programming algorithm with network cuts for solving the assembly line balancing problem", *Management Science*, Vol.30, No.1, 1984, pp.85-99.

98. Thangavelu, S.R. and C.M. Shetty, "Assembly line balancing by zero-one integer programming", *AIIE Transactions*, Vol.3, No.1, 1971, pp.61-68.

99. Thomopoulos, N.T., "Line balancing-sequencing for mixed-model assembly", *Management Science*, Vol.14, No.2, 1967, pp.B59-B75.

100. Thomopoulos, N.T., "Mixed model line balancing with smoothed station assignments", *Management Science*, Vol.16, No.9, 1970, pp.593-603.

101. Tonge, F.M., "Summary of a heuristic line balancing procedure", *Management Science*, Vol.7, No.1, 1960, pp.21-42.

102. Tonge, F.M., "A heuristic program for assembly line balancing", Prentice-Hall, Inc., New Jersey, 1961.

103. Tonge, F.M., "Assembly line balancing using probabilistic combinations of heuristics", *Management Science*, Vol.11, No.7, 1965, pp.727-735.

104. Townsend, W., "The single-machine problem with quadratic penalty function of completion times: A branch-and-bound solution", *Management Science*, Vol.24, No.5, 1978, pp.530-534.

105. Vrat, P., and A. Virani, "A cost model for optimal mix of balanced stochastic assembly line and the modular assembly system for a customer oriented production system", *International Journal of Production Research*, Vol.14, No.4, 1976, pp.445-463.

106. White, W.W., "Comments on a paper by Bowman", *Operations Research*, Vol.9, No.2, 1961, pp.274-276.

107. Wild, R., "Mass-production management", John Wiley & Sons, Ltd., New York, 1972.

108. Wild, R.,"On the selection of mass production systems", *International Journal of Production Research*, Vol.13, No.5, 1975, pp.443-461.

109. Wilhelm, W.E., "On the normality of operation times in small-lot assembly systems: a technial note", *International Journal of Production Research*, Vol.25, No.1, 1987, pp.145-149.

Appendix A. FORTRAN Code Of The Dynamic Programming Procedure

```
C
C    THIS PROGRAM SOLVES THE SINGLE-MODEL, STOCHASTIC ASSEMBLY
C    LINE BALANCING PROBLEM WITH THE DYNAMIC PROGRAMMING PROCEDURE
C    WITH THE BOUNDING STRATEGY
C
C    MAX. NUMBER OF TASKS < 50
C    MAX. TOTAL NUMBER OF STATE VARIABLES    < 15000
C    MAX. TOTAL NUMBER OF DECISION VARIABLES < 15000
C
C    VARIABLES OF THE PROGRAM:
C
C    BOUND     = VALUE OF ALPHA OF THE BOUNDING STRATEGY
C    COLUMN(I) = NUMBER OF THE COLUMN AT WHICH TASK I IS LOCATED ON
C             THE PRECEDENCE DIAGRAM
C    COST(I)   = TOTAL EXPECTED COST OF DECISION VARIABLE I
C    COSTS(I)  = TOTAL EXPECTED COST OF STATE VARIABLE I
C    CYCLE     = CYCLE TIME
C    IABORT    = 1 IF THE PROGRAM IS TERMINATED, OTHERWISE 0
C    IDEC(I,J) = IDENTITY OF THE JTH TASK IN DECISION VARIABLE I
C    IMFOL(I,J) = IDENTITY OF THE JTH. IMMEDIATE FOLLOWER OF TASK I
C    IMPRE(I,J) = IDENTITY OF THE JTH. IMMEDIATE PREDECESSOR OF TASK I
C    INCOM(I)  = INCOMPLETION COST OF TASK I
C    ISTA(I,J) = IDENTITY OF THE JTH TASK IN STATE VARIABLE I
C    MARK      = LIST OF TASKS AVAILABLE FOR ASSIGNMENT
C    MEAN(I)   = EXPECTED PROCESSING TIME OF TASK I
C    NCDEC     = COUNTER FOR THE NUMBER OF DECISION VARIABLES
C    NCOLUM    = NUMBER OF COLUMNS ON THE PRECEDENCE DIAGRAM
C    NCSTA     = COUNTER FOR THE NUMBER OF STATE VARIABLES
C    NDEC(I)   = NUMBER OF TASKS IN DECISION VARIABLE I
C    NFOL(I)   = NUMBER OF IMMEDIATE FOLLOWERS OF TASK I
C    NMARK     = NUMBER OF TASKS IN LIST "MARK"
C    NORMAL    = LIST WHICH CONTAINS CUMULATIVE NORMAL DISTRIBUTION
C    NPRE(I)   = NUMBER OF IMMEDIATE PREDECESSORS OF TASK I
C    NPREVD(I) = DECISION VARIABLE OF PREVIOUS STAGE ASSOCIATED WITH
C             DECISION VARIABLE I OF THE CURRENT STAGE
C    NPREVS(I) = STATE VARIABLE OF PREVIOUS STAGE ASSOCIATED WITH
C             STATE VARIABLE I OF THE CURRENT STAGE
C    NSTAGE    = CURRENT STAGE NUMBER
C    NTASK     = NUMBER OF TASKS IN THE PROBLEM
```

```
C     NTDEC(I)  = NUMBER OF DECISION VARIABLES OF STAGE I
C     NTSTA(I)  = NUMBER OF STATE VARIABLES OF STAGE I
C     PROB      = INCOMPLETION PROBABILITY OF A TASK
C     TIME      = CPU TIME SPENT IN SECONDS
C     TIMECK    = SUBROUTINE THAT DETERMINES THE AMOUNT OF CPU TIME
C              USED SINCE THE LAST CALL TO SUBROUTINE "TIMEON"
C     TIMEON    = SUBROUTINE THAT TURNS ON THE TIMER TO DETERMINE THE
C              AMOUNT OF CPU TIME USED
C     TOTIC(I)  = CUMULATIVE INCOMPLETION COST OF TASK I
C     VAR(I)    = VARIANVE OF THE PROCESSING TIME OF TASK I
C     WACY      = LABOR COST OF A STATION
C     WAGE      = LABOR RATE
C
C
C**** MAIN PROGRAM
C
C
      PARAMETER (LW=5,LTFP=20,LST=15000,LDE=15000,LDD=15000,LTASK=50)
      DIMENSION COSTS(LST),COST(LDE),VAR(LTASK),TOTIC(LTASK)
      REAL MEAN(LTASK),INCOM(LTASK),NORMAL(310),PRINC(LDE,LTASK)
      INTEGER*2 IGDEC(LDD,30),NTDEC(0:LTASK),NTSTA(LTASK),NSTA(LST),
     *ISTA(LST,LTASK),NDEC(LDE),IDEC(LDE,LTASK),NPREVS(LDE),
     *NPREVD(LST),IDUMY(LDD),MARK(LTASK),STATUS(LTASK),
     *NPRE(LTASK),IMPRE(LTASK,LTFP),COLUMN(LTASK),NFOL(LTASK),
     *IMFOL(LTASK,LTFP),NTPRE(LTASK),PREC(LTASK,LTASK),NTFOL(LTASK),
     *FOLLOW(LTASK,LTASK),IDMT(LTASK),ISDF(LTASK,LTASK)
      COMMON /MODEL/COST,COSTS,INCOM,MEAN,VAR,NORMAL,TOTIC,PRINC,PROB,
     *BOUND,WAGE,CYCLE,WACY,NTASK,NCDEC,NCOLUM,NDTOP,NSTAGE,NCSTA,IABORT,
     *ICEL,IKARA,IFLAG,STATUS,NTDEC,IDEC,MARK,NDEC,NPRE,IMPRE,ISTA,NSTA,
     *NTFOL,FOLLOW,ISDF,IDMT,NPREVS,NPREVD,IGDEC,IDUMY,IMFOL,NTSTA,
     *COLUMN,NFOL,NTPRE,PREC
C
      CALL TIMEON
      CALL INIT
      IF(IABORT.GT.0) GO TO 999
      JEND = 0
      CALL TIMECK(NTIME)
      TIME = NTIME / 100.
      WRITE(LW,950) TIME
      NMARK = 0
      DO 90 I=1,NTASK
      IF(NPRE(I).GT.0) GO TO 90
      NMARK = NMARK + 1
      MARK(NMARK) = I
   90 CONTINUE
      CALL DECVAR(NMARK,1,1)
      DO 110 I=1,NDTOP
  110 IGDEC(I,1) = 1
      NTDEC(1) = NDTOP
      NTSTA(1) = 1
      CALL TIMECK(NTIME)
      TIME = NTIME / 100.
      WRITE(LW,955) NSTAGE,TIME
      IF(TIME.GT.120.0) IABORT = 1
  100 CALL STATE
      IF(IABORT.GT.0) GO TO 999
      CALL REPORT(JEND)
      IF(JEND.NE.0) GO TO 120
      WRITE(LW,910) BOUND,NSTAGE
  120 IF(NSTAGE.EQ.NTASK) GO TO 990
      NSTAGE = NSTAGE + 1
      CALL DCSION
      IF(IABORT.GT.0) GO TO 999
      CALL TIMECK(NTIME)
      TIME = NTIME / 100.
      WRITE(LW,955) NSTAGE,TIME
      NCSTA = NCSTA + NTSTA(NSTAGE - 1)
      NCDEC = NCDEC + NTDEC(NSTAGE - 1)
      GO TO 100
  990 CONTINUE
      WRITE(LW,900)
      DO 995 I=1,NSTAGE
```

```
 995 WRITE(LW,905)I,NTSTA(I),NTDEC(I)
C
 999 STOP
 900 FORMAT(//,18X,'NUMBER  OF',6X,'NUMBER  OF',/,9X,'STAGE',4X,
    *'STATES',10X,'DECISION VARS.',/)
 905 FORMAT(10X,I2,8X,I4,14X,I4,/,8X,42('-'))
 910 FORMAT(//,4X,'ALPHA VALUE OF ',F7.3,' IS TOO LOW TO FORM A '
    *,I3,'-STATION DESIGN')
 950 FORMAT(/,4X,'CPU SPENT FOR READING DATA AND INITIALIZATION = ',
    *F7.3,' SECONDS')
 955 FORMAT(/,4X,'TOTAL CPU SPENT AT THE END OF STAGE',I3,' = ',
    *F7.3,' SECONDS')
     END
C
C**** SUBROUTINE "INIT" READS DATA, INITIALIZES VARIABLES AND COMPUTES
C**** THE CUMULATIVE INCOMPLETION COSTS OF THE TASKS
C
     SUBROUTINE INIT
     PARAMETER (LW=5,LRD=4,LRN=3,LST=15000,LDE=15000,LDD=15000,
    *LTFP=20,LTASK=50)
     DIMENSION COSTS(LST),COST(LDE),VAR(LTASK),TOTIC(LTASK)
     REAL MEAN(LTASK),INCOM(LTASK),NORMAL(310),PRINC(LDE,LTASK)
     INTEGER*2 IGDEC(LDD,30),NTDEC(0:LTASK),NTSTA(LTASK),NSTA(LST),
    *ISTA(LST,LTASK),NDEC(LDE),IDEC(LDE,LTASK),NPREVS(LDE),
    *NPREVD(LST),IDUMY(LDD),MARK(LTASK),STATUS(LTASK),
    *NPRE(LTASK),IMPRE(LTASK,LTFP),COLUMN(LTASK),NFOL(LTASK),
    *IMFOL(LTASK,LTFP),NTPRE(LTASK),PREC(LTASK,LTASK),
    *NTFOL(LTASK),FOLLOW(LTASK,LTASK),IDMT(LTASK),ISDF(LTASK,LTASK)
     COMMON /MODEL/COST,COSTS,INCOM,MEAN,VAR,NORMAL,TOTIC,PRINC,PROB,
    *BOUND,WAGE,CYCLE,WACY,NTASK,NCDEC,NCOLUM,NDTOP,NSTAGE,NCSTA,IABORT,
    *ICEL,IKARA,IFLAG,STATUS,NTDEC,IDEC,MARK,NDEC,NPRE,IMPRE,ISTA,NSTA,
    *NTFOL,FOLLOW,ISDF,IDMT,NPREVS,NPREVD,IGDEC,IDUMY,IMFOL,NTSTA,
    *COLUMN,NFOL,NTPRE,PREC
C
C   READ THE DATA FROM FILES 3 AND 4
C
     READ(LRD,*,END=990) IFLAG
     IF(IFLAG.EQ.0)  GO TO 5
     READ(LRD,*,END=990) ICEL,WAGE,CYCLE,IKARA,RAN1,RAN2,BOUND
     DO 201 I=1,ICEL
     READ(LRD,*,END=990) MEAN(I),COLUMN(I),NFOL(I),NPRE(I)
     IF(NFOL(I).EQ.0) GO TO 211
     READ(LRD,*,END=990) (IMFOL(I,J),J=1,NFOL(I))
 211 IF(NPRE(I).EQ.0) GO TO 201
     READ(LRD,*,END=990) (IMPRE(I,J),J=1,NPRE(I))
 201 CONTINUE
     DO 79 I=1,ICEL
     IDMT(I) = 0
     INCOM(I) = RAN2 * MEAN(I)
  79 CONTINUE
     DO 601 I=1,ICEL
     K = COLUMN(I)
     IF(K.EQ.IKARA)  GO TO 601
     IF(NFOL(I).EQ.0) GO TO 601
     DO 611 J=1,ICEL
 611 STATUS(J) = 0
     DO 621 J=1,NFOL(I)
 621 STATUS(IMFOL(I,J)) = 1
     DO 631 N=K,IKARA
     DO 641 J=1,ICEL
     IF(COLUMN(J).NE.N)  GO TO 641
     IF(STATUS(J).EQ.0)  GO TO 641
     IF(NFOL(J).EQ.0)    GO TO 641
     DO 651 IB=1,NFOL(J)
 651 STATUS(IMFOL(J,IB)) = 1
 641 CONTINUE
 631 CONTINUE
     DO 661 J=1,ICEL
     IF(STATUS(J).LT.1)  GO TO 661
     IDMT(I) = IDMT(I) + 1
     ISDF(I,IDMT(I)) = J
 661 CONTINUE
```

```
  601 CONTINUE
C
    5 READ(LRD,*,END=990) NTASK,WAGE,CYCLE,NCOLUM,RAN1,RAN2,BOUND
      DO 200 I=1,NTASK
      READ(LRD,*,END=990) MEAN(I),COLUMN(I),NFOL(I),NPRE(I)
      IF(NFOL(I).EQ.0) GO TO 210
      READ(LRD,*,END=990) (IMFOL(I,J),J=1,NFOL(I))
  210 IF(NPRE(I).EQ.0) GO TO 200
      READ(LRD,*,END=990) (IMPRE(I,J),J=1,NPRE(I))
  200 CONTINUE
      READ(LRN,*,END=990) (NORMAL(I),I=1,310)
C
      WACY = WAGE * (CYCLE / 60.)
      DO 89 I=1,NTASK
   89 INCOM(I) = RAN2 * MEAN(I)
      DO 90 I=1,NTASK
      DO 85 J=1,NTASK
   85 STATUS(J) = 0
      STATUS(I) = 1
      K = COLUMN(I)
      DO 75 N=K,NCOLUM
      DO 70 J=1,NTASK
      IF(COLUMN(J).NE.N)  GO TO 70
      IF(STATUS(J).NE.1)  GO TO 70
      IF(NFOL(J).EQ.0)    GO TO 70
      IA = NFOL(J)
      DO 65 IB=1,IA
   65 STATUS(IMFOL(J,IB)) = 1
   70 CONTINUE
   75 CONTINUE
      TOTIC(I) = 0.0
      DO 60 J=1,NTASK
      IF(STATUS(J).LT.1)  GO TO 60
      TOTIC(I) = TOTIC(I) + INCOM(J)
   60 CONTINUE
   90 CONTINUE
C
      DO 92 I=1,NTASK
      DUMMM = RAN1  * MEAN(I)
   92 VAR(I) = DUMMM * DUMMM
      DO 189 I=1,NTASK
      NTFOL(I) = 0
  189 NTPRE(I) = 0
      DO 190 I=1,NTASK
      K = COLUMN(I) - 1
      IF(K.EQ.0)  GO TO 190
      DO 185 J=1,I
      STATUS(J) = 0
  185 CONTINUE
      DO 186 J=1,NPRE(I)
      STATUS(IMPRE(I,J)) = 1
  186 CONTINUE
      DO 175 N=1,K
      M = K - N + 1
      DO 170 J=1,I
      IF(COLUMN(J).NE.M)  GO TO 170
      IF(STATUS(J).EQ.0)  GO TO 170
      IF(NPRE(J).EQ.0)  GO TO 170
      IA = NPRE(J)
      DO 165 IB=1,IA
  165 STATUS(IMPRE(J,IB)) = 1
  170 CONTINUE
  175 CONTINUE
      DO 160 J=1,I
      IF(STATUS(J).LT.1)  GO TO 160
      NTPRE(I) = NTPRE(I) + 1
      PREC(I,NTPRE(I)) = J
  160 CONTINUE
  190 CONTINUE
C
      DO 600 I=1,NTASK
      K = COLUMN(I)
```

```
     IF(K.EQ.NCOLUM)  GO TO 600
     IF(NFOL(I).EQ.0) GO TO 600
     DO 610 J = 1,NTASK
     STATUS(J) = 0
 610 CONTINUE
     DO 620 J = 1,NFOL(I)
     STATUS(IMFOL(I,J)) = 1
 620 CONTINUE
     DO 630 N = K,NCOLUM
     DO 640 J = 1,NTASK
     IF(COLUMN(J).NE.N)  GO TO 640
     IF(STATUS(J).EQ.0)  GO TO 640
     IF(NFOL(J).EQ.0)    GO TO 640
     DO 650 IB = 1,NFOL(J)
     STATUS(IMFOL(J,IB)) = 1
 650 CONTINUE
 640 CONTINUE
 630 CONTINUE
     DO 660 J = 1,NTASK
     IF(STATUS(J).LT.1)  GO TO 660
     NTFOL(I) = NTFOL(I) + 1
     FOLLOW(I,NTFOL(I)) = J
 660 CONTINUE
 600 CONTINUE
C
     DO 100 J = 1,LDD
     DO 100 K = 1,30
     IGDEC(J,K) = 0
 100 CONTINUE
     DO 110 J = 1,LTASK
     NTSTA(J) = 0
     NTDEC(J) = 0
     MARK(J) = 0
 110 STATUS(J) = 0
     DO 120 J = 1,LDE
     NDEC(J) = 0
     NPREVS(J) = 0
 120 COST(J) = 0.0
     DO 125 J = 1,LST
     NSTA(J) = 0
     NPREVD(J) = 0
     COSTS(J) = 0.0
     DO 125 JJ = 1,LTASK
 125 PRINC(J,JJ) = 0.0
     IABORT = 0
     NTDEC(0) = 0
     NCDEC = 0
     NCSTA = 0
     NSTAGE = 1
     WRITE(LW,900) CYCLE,WAGE,BOUND
     WRITE(LW,905)
     DO 300 I = 1,NTASK
     IF(NFOL(I).EQ.0) GO TO 310
     WRITE(LW,910) I,MEAN(I),VAR(I),INCOM(I),TOTIC(I),
    *(IMFOL(I,J),J = 1,NFOL(I))
     GO TO 300
 310 WRITE(LW,915) I,MEAN(I),VAR(I),INCOM(I),TOTIC(I)
 300 CONTINUE
     RETURN
C
 990 IABORT = 1
     WRITE(LW,920)
     RETURN
C
 900 FORMAT(//,9X,'CYCLE TIME = ',F5.1,2X,'MINUTES',/,9X,'LABOR RATE = ',
    *F5.2,2X,'$/HOUR',/,9X,'BOUND      = ',F5.3,//)
 905 FORMAT(//,28X,'INCOMPLETION   CUMULATIVE   IMMEDIATE',/,4X,'TASK',
    *3X,'MEAN',3X,'VARIANCE',4X,'COST',13X,'COST',6X,'FOLLOWERS',/)
 910 FORMAT(5X,I2,3X,F5.2,4X,F6.3,5X,F6.3,8X,F6.3,6X,5(I2,','))
 915 FORMAT(5X,I2,3X,F5.2,4X,F6.3,5X,F6.3,8X,F6.3,6X,'NONE')
 920 FORMAT(5X,'ERROR IN INPUT DATA FILES, LESS DATA THAN EXPECTED')
```

```
      END
C
C**** SUBROUTINE 'DECVAR' GENERATES THE COMBINATIONS OF THE TASKS
C**** IN LIST 'MARK'
C
      SUBROUTINE DECVAR(NMARK,I,MT1)
      PARAMETER (LW=5,LTFP=20,LST=15000,LDE=15000,LDD=15000,LTASK=50)
      DIMENSION COSTS(LST),COST(LDE),VAR(LTASK),TOTIC(LTASK)
      REAL MEAN(LTASK),INCOM(LTASK),NORMAL(310),PRINC(LDE,LTASK)
      INTEGER*2 IGDEC(LDD,30),NTDEC(0:LTASK),NTSTA(LTASK),NSTA(LST),
     *ISTA(LST,LTASK),NDEC(LDE),IDEC(LDE,LTASK),NPREVS(LDE),
     *NPREVD(LST),IDUMY(LDD),MARK(LTASK),STATUS(LTASK),
     *NPRE(LTASK),IMPRE(LTASK,LTFP),COLUMN(LTASK),NFOL(LTASK),
     *IMFOL(LTASK,LTFP),NEL(15,525),NS(0:15),IFS(15,525,10),
     *NIFS(15,525),MA(LTASK),JA(LTASK),NTPRE(LTASK),PREC(LTASK,LTASK),
     *NTFOL(LTASK),FOLLOW(LTASK,LTASK),IDMT(LTASK),ISDF(LTASK,LTASK)
      COMMON /MODEL/COST,COSTS,INCOM,MEAN,VAR,NORMAL,TOTIC,PRINC,PROB,
     *BOUND,WAGE,CYCLE,WACY,NTASK,NCDEC,NCOLUM,NDTOP,NSTAGE,NCSTA,IABORT,
     *ICEL,IKARA,IFLAG,STATUS,NTDEC,IDEC,MARK,NDEC,NPRE,IMPRE,ISTA,NSTA,
     *NTFOL,FOLLOW,ISDF,IDMT,NPREVS,NPREVD,IGDEC,IDUMY,IMFOL,NTSTA,
     *COLUMN,NFOL,NTPRE,PREC
C
      LINS = NCDEC + NTDEC(NSTAGE - 1) + I
      NS(0) = 0
      ITEST = LDE - 5
      CALL TIMECK(NTIME)
      IF(NTIME.GT.12000) GO TO 9000
      DO 100 J=1,NTASK
      JA(J) = 0
      IF(STATUS(J).EQ.1) THEN
      MA(J) = 1
      ELSE
      MA(J) = 0
      ENDIF
  100 CONTINUE
      DO 110 J=1,NMARK
  110 MA(MARK(J)) = 1
      INDF = (2 ** NMARK) - 1
      LINE = 1
      KSTA = 1
      DO 120 L=1,INDF
      M = 0
      DO 130 J=1,NMARK
      IF(MATRIX(L,J).EQ.0) GO TO 130
      M = M + 1
      IDEC(LINS,M) = MARK(J)
  130 CONTINUE
      CALL PRCA(LINS,M)
C**** IF INCOMPLETION PROBABILITY IS LARGER THAN BOUND,
C**** THE DECISION VARIABLE IS NOT GENERATED
      IF(PROB.GT.BOUND) GO TO 140
      NEL(KSTA,LINE) = M
      NDEC(LINS) = M
      CALL COSCAL(LINS,M,MT1)
      LINS = LINS + 1
      LINE = LINE + 1
      IF(LINS.GT.ITEST) GO TO 800
      GO TO 120
  140 CONTINUE
      DO 150 N=1,M
  150 IDEC(LINS,N) = 0
      NEL(KSTA,LINE) = 0
      NDEC(LINS) = 0
  120 CONTINUE
      NS(KSTA) = LINE - 1
C
C**** FOLLOWING PART FINDS THE ASSOCIATED UNMARKED IMMEDIATE FOLLOWERS
C
  400 N = NS(KSTA)
      LINF = LINS - N - 1
      DO 160 L=1,N
      K = 1
      LINF = LINF + 1
      DO 170 J = 1,NTASK
      IA = 0
      IF(MA(J).EQ.1) GO TO 170
      IF(STATUS(J).EQ.1) GO TO 170
      IB = NPRE(J)
      DO 180 J1 = 1,IB
      IF(STATUS(IMPRE(J,J1)).EQ.1) GO TO 190
      IC = NEL(KSTA,L)
      DO 200 J2 = 1,IC
      IF(IMPRE(J,J1).EQ.IDEC(LINF,J2)) GO TO 190
  200 CONTINUE
      GO TO 180
  190 IA = IA + 1
  180 CONTINUE
      IF(IA.NE.IB) GO TO 170
      IFS(KSTA,L,K) = J
      JA(J) = 1
      K = K + 1
  170 CONTINUE
      NIFS(KSTA,L) = K - 1
  160 CONTINUE
C
      DO 210 L = 1,NTASK
      IF(JA(L).EQ.1) MA(L) = 1
  210 CONTINUE
      NDUM = 0
      LINE = 0
      LINF = LINS - NS(KSTA) - 1
      DO 220 L = 1,N
      LINF = LINF + 1
      IF(NIFS(KSTA,L).EQ.0) GO TO 220
      NDUM = NDUM + 1
      IKF = (2 ** (NIFS(KSTA,L))) - 1
      DO 230 IC = 1,IKF
      IA = NEL(KSTA,L)
      DO 240 L1 = 1,IA
  240 IDEC(LINS,L1) = IDEC(LINF,L1)
      M = NEL(KSTA,L)
      IB = NIFS(KSTA,L)
      DO 250 J = 1,IB
      IF(MATRIX(IC,J).EQ.0) GO TO 250
      M = M + 1
      IDEC(LINS,M) = IFS(KSTA,L,J)
  250 CONTINUE
      CALL PRCA(LINS,M)
      IF(PROB.GT.BOUND) GO TO 260
      LINE = LINE + 1
      NEL(KSTA + 1,LINE) = M
      NDEC(LINS) = M
      CALL COSCAL(LINS,M,MT1)
      LINS = LINS + 1
      IF(LINS.GT.ITEST) GO TO 800
      GO TO 230
  260 CONTINUE
      DO 270 J6 = 1,M
  270 IDEC(LINS,J6) = 0
      NEL(KSTA + 1,LINE) = 0
      NDEC(LINS) = 0
  230 CONTINUE
  220 CONTINUE
      IF(NDUM.EQ.0) GO TO 500
      KSTA = KSTA + 1
      NS(KSTA) = LINE
      GO TO 400
  500 NDTOP = LINS - NCDEC - NTDEC(NSTAGE - 1) - I
      RETURN
  800 IABORT = 1
      WRITE(LW,900)
      RETURN
 9000 IABORT = 1
      WRITE(LW,9010)
      RETURN
  900 FORMAT(3X,'# OF STATES EXCEEDED LIMIT IN SUBROUTINE 'DECVAR"')
 9010 FORMAT(3X,'CPU TIME LIMIT IS EXCEEDED')
      END
C
C**** SUBROUTINE 'COSCAL' CALCULATES INCOMPLETION COSTS
C**** OF THE DECISION VARIABLES
C
      SUBROUTINE COSCAL(K,M,MT1)
      PARAMETER (LW=5,LTFP=20,LST=15000,LDE=15000,LDD=15000,LTASK=50)
      DIMENSION COSTS(LST),COST(LDE),VAR(LTASK),TOTIC(LTASK),
     *PRINC(LDE,LTASK)
      REAL MEAN(LTASK),INCOM(LTASK),NORMAL(310)
      INTEGER*2 IGDEC(LDD,30),NTDEC(0:LTASK),NTSTA(LTASK),NSTA(LST),
     *ISTA(LST,LTASK),NDEC(LDE),IDEC(LDE,LTASK),NPREVS(LDE),
     *NPREVD(LST),IDUMY(LDD),MARK(LTASK),STATUS(LTASK),
     *NPRE(LTASK),IMPRE(LTASK,LTFP),COLUMN(LTASK),NFOL(LTASK),
     *IMFOL(LTASK,LTFP),NTPRE(LTASK),PREC(LTASK,LTASK),COM(LTASK),
     *NTFOL(LTASK),FOLLOW(LTASK,LTASK),NST(LTASK),LABEL(LTASK),
     *FINISH(LTASK),NSTART(LTASK,LTASK),BNO,FINO,IDMT(LTASK),
     *ISDF(LTASK,LTASK)
      COMMON /MODEL/COST,COSTS,INCOM,MEAN,VAR,NORMAL,TOTIC,PRINC,PROB,
     *BOUND,WAGE,CYCLE,WACY,NTASK,NCDEC,NCOLUM,NDTOP,NSTAGE,NCSTA,IABORT,
     *ICEL,IKARA,IFLAG,STATUS,NTDEC,IDEC,MARK,NDEC,NPRE,IMPRE,ISTA,NSTA,
     *NTFOL,FOLLOW,ISDF,IDMT,NPREVS,NPREVD,IGDEC,IDUMY,IMFOL,NTSTA,
     *COLUMN,NFOL,NTPRE,PREC
      COMMON /EKLEME/BNO,FINO,NST,NSTART,FINISH
C
      DO 90 L=1,M
      IDKL = IDEC(K,L)
      CPAR = 0.0
      IF(L.EQ.1) GO TO 100
      LMNS1 = L - 1
      INDF = 2 ** LMNS1
      I = 1
  105 CONTINUE
      DO 120 J=1,LMNS1
      IF(MATCOM(I,J).EQ.1)  THEN
      COM(J) = 1
      ELSE
      COM(J) = 0
      ENDIF
  120 CONTINUE
      COM(L) = 0
      FINO = 0
      DO 130 J=1,L
      IF(COM(J).EQ.1)  GO TO 130
      IDKJ = IDEC(K,J)
      DO 150 IA=1,NTASK
      DO 160 IG=1,J
      IF(IA.EQ.IDEC(K,IG))  GO TO 150
  160 CONTINUE
      IF(FINO.EQ.0)  GO TO 156
      DO 155 IH=1,FINO
      IF(IA.EQ.FINISH(IH))  GO TO 150
  155 CONTINUE
  156 DO 170 IG=1,NTPRE(IDKJ)
      IF(PREC(IDKJ,IG).NE.IA)  GO TO 170
      FINO = FINO + 1
      FINISH(FINO) = IA
      GO TO 150
  170 CONTINUE
  150 CONTINUE
  130 CONTINUE
  987 BNO = 0
      DO 200 J=1,LMNS1
      IF(COM(J).EQ.0)  GO TO 200
      IDKJ = IDEC(K,J)
      IF(NTPRE(IDKJ).EQ.0)  GO TO 110
      BNO = BNO + 1
      NST(BNO) = 0
      DO 210 IA=1,NTASK
```

```
    DO 220 IB=1,J
    IF(IA.EQ.IDEC(K,IB))  GO TO 210
220 CONTINUE
    DO 230 IB=1,FINO
    IF(IA.EQ.FINISH(IB))  GO TO 210
230 CONTINUE
    IF(NST(BNO).EQ.0)  GO TO 245
    DO 246 IH=1,NST(BNO)
    IF(IA.EQ.NSTART(BNO,IH))  GO TO 210
246 CONTINUE
245 DO 240 IB=1,NTPRE(IDKJ)
    IF(PREC(IDKJ,IB).NE.IA)  GO TO 240
    NST(BNO) = NST(BNO) + 1
    NSTART(BNO,NST(BNO)) = IA
240 CONTINUE
210 CONTINUE
    IF(NST(BNO).EQ.0)  GO TO 110
200 CONTINUE
    GO TO 400
100 FINO = 0
    BNO = 0
    COM(L) = 0
    IF(NTPRE(IDKL).EQ.0)  GO TO 400
    DO 300 IA=1,NTASK
    DO 310 IB=1,NTPRE(IDKL)
    IF(PREC(IDKL,IB).NE.IA)  GO TO 310
    FINO = FINO + 1
    FINISH(FINO) = IA
    GO TO 300
310 CONTINUE
300 CONTINUE
400 IF(NSTAGE.GT.1)  GO TO 410
    SANCOM = 1.0
    GO TO 420
410 CALL PRCMB(MT1,K,SANCOM)
420 TOPLM = 0.0
    TOPLV = 0.0
    DO 415 J=1,L
    IF(COM(J).EQ.1)  GO TO 415
    TOPLMO = TOPLM
    TOPLVO = TOPLV
    TOPLM = TOPLM + MEAN(IDEC(K,J))
    TOPLV = TOPLV + VAR(IDEC(K,J))
415 CONTINUE
    IF(TOPLMO.GT.0.0)  GO TO 604
    SANONC = 0.0
    GO TO 605
604 FNORMO = (CYCLE - TOPLMO) / SQRT(TOPLVO)
    CALL PRCA2(SANONC,FNORMO)
605 FNORM = (CYCLE - TOPLM) / SQRT(TOPLV)
    CALL PRCA2(SANTAK,FNORM)
    SANGEC = (SANTAK - SANONC) * SANCOM
    PRINC(K,L) = PRINC(K,L) + SANGEC
    IS = MT1
    KC = K
    DO 550 IA=1,NSTAGE
    IF(IA.EQ.1)  GO TO 555
    DO 560 IB=1,NDEC(KC)
    CTOP = 0.0
    IDKCIB = IDEC(KC,IB)
    DO 565 IG=1,FINO
    IF(IDKCIB.EQ.FINISH(IG))  GO TO 560
565 CONTINUE
    DO 570 IG=1,NTASK
    LABEL(IG) = 0
570 CONTINUE
    DO 585 IXY=IB,NDEC(KC)
    IDX = IDEC(KC,IXY)
    DO 580 IG=1,NTFOL(IDX)
580 LABEL(FOLLOW(IDX,IG)) = 1
585 CONTINUE
    DO 591 IXY=L,M
```

```
   IDX = IDEC(K,IXY)
   DO 586 IG=1,NTFOL(IDX)
   IF(LABEL(FOLLOW(IDX,IG)).EQ.1) LABEL(FOLLOW(IDX,IG))=2
 586 CONTINUE
 591 CONTINUE
   DO 592 IXY=1,NTASK
   IF(LABEL(IXY).EQ.2) CTOP = CTOP + INCOM(IXY)
 592 CONTINUE
   CPAR = CPAR + (CTOP * PRINC(KC,IB) * SANGEC)
 560 CONTINUE
 555 KC = NPREVD(IS)
   IS = NPREVS(IS)
 550 CONTINUE
 416 IF(L.EQ.1) GO TO 108
 110 I = I + 1
   IF(I.LE.INDF) GO TO 105
 108 CONTINUE
   IF(IFLAG.EQ.0) THEN
   DO 600 I=1,NTASK
 600 LABEL(I) = 0
   DO 610 I=L,M
   IDKI = IDEC(K,I)
   LABEL(IDKI) = 1
   DO 620 IA=1,NTFOL(IDKI)
 620 LABEL(FOLLOW(IDKI,IA)) = 1
 610 CONTINUE
   CIC = 0.0
   DO 630 I=1,NTASK
   IF(LABEL(I).EQ.1) CIC = CIC + INCOM(I)
 630 CONTINUE
   COST(K) = COST(K) + (PRINC(K,L) * CIC) - CPAR
   ELSE
   DO 601 I=1,ICEL
 601 LABEL(I) = 0
   DO 611 I=L,M
   IDKI = IDEC(K,I)
   LABEL(IDKI) = 1
   DO 621 IA=1,IDMT(IDKI)
 621 LABEL(ISDF(IDKI,IA)) = 1
 611 CONTINUE
   CIC = 0.0
   DO 631 I=1,ICEL
   IF(LABEL(I).EQ.1) CIC = CIC + INCOM(I)
 631 CONTINUE
   COST(K) = COST(K) + (PRINC(K,L) * CIC) - CPAR
   ENDIF
  90 CONTINUE
   COST(K) = COST(K) + WACY
   RETURN
   END
C
C**** SUBROUTINE "PRCA" CALCULATES INCOMPLETION PROBABILITIES
C**** OF THE TASKS
C
   SUBROUTINE PRCA(N,M)
   PARAMETER (LW=5,LTFP=20,LST=15000,LDE=15000,LDD=15000,LTASK=50)
   DIMENSION COSTS(LST),COST(LDE),VAR(LTASK),TOTIC(LTASK)
   REAL MEAN(LTASK),INCOM(LTASK),NORMAL(310),PRINC(LDE,LTASK)
   INTEGER*2 IGDEC(LDD,30),NTDEC(0:LTASK),NTSTA(LTASK),NSTA(LST),
  *ISTA(LST,LTASK),NDEC(LDE),IDEC(LDE,LTASK),NPREVS(LDE),
  *NPREVD(LST),IDUMY(LDD),MARK(LTASK),STATUS(LTASK),
  *NPRE(LTASK),IMPRE(LTASK,LTFP),COLUMN(LTASK),NFOL(LTASK),
  *IMFOL(LTASK,LTFP),NTPRE(LTASK),PREC(LTASK,LTASK),
  *NTFOL(LTASK),FOLLOW(LTASK,LTASK),IDMT(LTASK),ISDF(LTASK,LTASK)
   COMMON /MODEL/COST,COSTS,INCOM,MEAN,VAR,NORMAL,TOTIC,PRINC,PROB,
  *BOUND,WAGE,CYCLE,WACY,NTASK,NCDEC,NCOLUM,NDTOP,NSTAGE,NCSTA,IABORT,
  *ICEL,IKARA,IFLAG,STATUS,NTDEC,IDEC,MARK,NDEC,NPRE,IMPRE,ISTA,NSTA,
  *NTFOL,FOLLOW,ISDF,IDMT,NPREVS,NPREVD,IGDEC,IDUMY,IMFOL,NTSTA,
  *COLUMN,NFOL,NTPRE,PREC
C
   TMEAN = 0.0
   TVAR = 0.0
```

```
      DO 100 J=1,M
      TMEAN = TMEAN + MEAN(IDEC(N,J))
  100 TVAR = TVAR + VAR(IDEC(N,J))
      FNORM = (CYCLE - TMEAN) / SQRT(TVAR)
      IF(FNORM) 110,120,120
  110 IF(FNORM.LT.-3.0) GO TO 130
      J = (0.01 - FNORM) * 100.
      PROB = NORMAL(J)
      RETURN
  130 PROB = 1.
      RETURN
  120 IF(FNORM.GT.3.0) GO TO 140
      J = (FNORM + 0.01) * 100.
      PROB = 1. - NORMAL(J)
      RETURN
  140 PROB = 0.0
      RETURN
      END
C
C**** SUBROUTINE "STATE" GENERATES STATE VARIABLES OF THE CURRENT STAGE
C
      SUBROUTINE STATE
      PARAMETER (LW=5,LTFP=20,LST=15000,LDE=15000,LDD=15000,LTASK=50)
      DIMENSION COSTS(LST),COST(LDE),VAR(LTASK),TOTIC(LTASK)
      REAL MEAN(LTASK),INCOM(LTASK),NORMAL(310),PRINC(LDE,LTASK)
      INTEGER*2 IGDEC(LDD,30),NTDEC(0:LTASK),NTSTA(LTASK),NSTA(LST),
     *ISTA(LST,LTASK),NDEC(LDE),IDEC(LDE,LTASK),NPREVS(LDE),
     *NPREVD(LST),IDUMY(LDD),MARK(LTASK),STATUS(LTASK),
     *NPRE(LTASK),IMPRE(LTASK,LTFP),COLUMN(LTASK),NFOL(LTASK),
     *IMFOL(LTASK,LTFP),NTPRE(LTASK),PREC(LTASK,LTASK),
     *NTFOL(LTASK),FOLLOW(LTASK,LTASK),IDMT(LTASK),ISDF(LTASK,LTASK)
      COMMON /MODEL/COST,COSTS,INCOM,MEAN,VAR,NORMAL,TOTIC,PRINC,PROB,
     *BOUND,WAGE,CYCLE,WACY,NTASK,NCDEC,NCOLUM,NDTOP,NSTAGE,NCSTA,IABORT,
     *ICEL,IKARA,IFLAG,STATUS,NTDEC,IDEC,MARK,NDEC,NPRE,IMPRE,ISTA,NSTA,
     *NTFOL,FOLLOW,ISDF,IDMT,NPREVS,NPREVD,IGDEC,IDUMY,IMFOL,NTSTA,
     *COLUMN,NFOL,NTPRE,PREC
C
      I = 1
      IA = NTSTA(NSTAGE)
      IB = NTDEC(NSTAGE)
      ITEST = LST - 5
      DO 100 JA=1,IA
      DO 101 JB=1,IB
      IC = NCSTA + JA
      ID = NCDEC + JB
      IE = NSTA(IC)
      IF = NCSTA + NTSTA(NSTAGE) + I
      NCTASK = 0
      IF(IF.GT.ITEST) GO TO 800
      DO 102 JC=1,30
      IF(IGDEC(JB,JC).EQ.JA) GO TO 155
  102 CONTINUE
      GO TO 101
  155 IF(NSTA(NCSTA + JA).EQ.0) GO TO 99
      DO 103 JC=1,IE
      NCTASK = NCTASK + 1
  103 ISTA(IF,NCTASK) = ISTA(IC,JC)
   99 IG = NDEC(ID)
      DO 104 JC=1,IG
      NCTASK = NCTASK + 1
  104 ISTA(IF,NCTASK) = IDEC(ID,JC)
      NSTA(IF) = NCTASK
      COSTS(IF) = COSTS(IC) + COST(ID)
      NPREVS(IF) = IC
      NPREVD(IF) = ID
      I = I + 1
  101 CONTINUE
      NTSTA(NSTAGE + 1) = I - 1
      IF(JA.EQ.1) GO TO 100
      CALL STAELM(I)
  100 CONTINUE
      NTSTA(NSTAGE + 1) = I - 1
      RETURN
 800  IABORT = 1
      WRITE(LW,900)
 900  FORMAT(3X,'# OF STATES EXCEEDED LIMIT IN SUBROUTINE "STATE"')
      RETURN
      END
C
C**** SUBROUTINE "STAELM" ELIMINATES THE IDENTICAL STATE VARIABLES
C
      SUBROUTINE STAELM(I)
      PARAMETER (LW=5,LTFP=20,LST=15000,LDE=15000,LDD=15000,LTASK=50)
      DIMENSION COSTS(LST),COST(LDE),VAR(LTASK),TOTIC(LTASK)
      REAL MEAN(LTASK),INCOM(LTASK),NORMAL(310),PRINC(LDE,LTASK)
      INTEGER*2 IGDEC(LDD,30),NTDEC(0:LTASK),NTSTA(LTASK),NSTA(LST),
     *ISTA(LST,LTASK),NDEC(LDE),IDEC(LDE,LTASK),NPREVS(LDE),
     *NPREVD(LST),IDUMY(LDD),MARK(LTASK),STATUS(LTASK),
     *NPRE(LTASK),IMPRE(LTASK,LTFP),COLUMN(LTASK),NFOL(LTASK),
     *IMFOL(LTASK,LTFP),NTPRE(LTASK),PREC(LTASK,LTASK),
     *NTFOL(LTASK),FOLLOW(LTASK,LTASK),IDMT(LTASK),ISDF(LTASK,LTASK)
      COMMON /MODEL/COST,COSTS,INCOM,MEAN,VAR,NORMAL,TOTIC,PRINC,PROB,
     *BOUND,WAGE,CYCLE,WACY,NTASK,NCDEC,NCOLUM,NDTOP,NSTAGE,NCSTA,IABORT,
     *ICEL,IKARA,IFLAG,STATUS,NTDEC,IDEC,MARK,NDEC,NPRE,IMPRE,ISTA,NSTA,
     *NTFOL,FOLLOW,ISDF,IDMT,NPREVS,NPREVD,IGDEC,IDUMY,IMFOL,NTSTA,
     *COLUMN,NFOL,NTPRE,PREC
C
      J = I - 1
      DO 100 K=1,J
  100 IDUMY(K) = 1
      K = I - 2
      DO 110 L=1,K
      IF(IDUMY(L).EQ.0) GO TO 110
      M = L + 1
      DO 120 N=M,J
      IA = NCSTA + NTSTA(NSTAGE) + L
      IB = NCSTA + NTSTA(NSTAGE) + N
      IF(NSTA(IA).NE.NSTA(IB)) GO TO 120
      IC = NSTA(IA)
      DO 130 ID=1,IC
      DO 140 IE=1,IC
      IF((ISTA(IA,ID)).EQ.(ISTA(IB,IE))) GO TO 130
  140 CONTINUE
      GO TO 120
  130 CONTINUE
      IF(COSTS(IA).GT.COSTS(IB)) GO TO 150
      IDUMY(L) = 1
      IDUMY(N) = 0
  120 CONTINUE
      IDUMY(L) = 1
      GO TO 110
  150 IDUMY(L) = 0
  110 CONTINUE
      I = 1
      DO 160 L=1,J
      IF((IDUMY(L)).EQ.0) GO TO 160
      IA = NCSTA + NTSTA(NSTAGE) + L
      IB = NSTA(IA)
      IC = NCSTA + NTSTA(NSTAGE) + I
      DO 170 M=1,IB
  170 ISTA(IC,M) = ISTA(IA,M)
      NSTA(IC) = NSTA(IA)
      COSTS(IC) = COSTS(IA)
      NPREVS(IC) = NPREVS(IA)
      NPREVD(IC) = NPREVD(IA)
      I = I + 1
  160 CONTINUE
      RETURN
      END
C
C**** SUBROUTINE "DCSION" GENERATES DECISION VARIABLES CORRESPONDING
C**** TO A STATE VARIABLE
C
C
```

```
      SUBROUTINE DCSION
      PARAMETER (LW=5,LTFP=20,LST=15000,LDE=15000,LDD=15000,LTASK=50)
      DIMENSION COSTS(LST),COST(LDE),VAR(LTASK),TOTIC(LTASK)
      REAL MEAN(LTASK),INCOM(LTASK),NORMAL(310),PRINC(LDE,LTASK)
      INTEGER*2 IGDEC(LDD,30),NTDEC(0:LTASK),NTSTA(LTASK),NSTA(LST),
     *ISTA(LST,LTASK),NDEC(LDE),IDEC(LDE,LTASK),NPREVS(LDE),
     *NPREVD(LST),IDUMY(LDD),MARK(LTASK),STATUS(LTASK),
     *NPRE(LTASK),IMPRE(LTASK,LTFP),COLUMN(LTASK),NFOL(LTASK),
     *IMFOL(LTASK,LTFP),NTPRE(LTASK),PREC(LTASK,LTASK),
     *NTFOL(LTASK),FOLLOW(LTASK,LTASK),IDMT(LTASK),ISDF(LTASK,LTASK)
      COMMON /MODEL/COST,COSTS,INCOM,MEAN,VAR,NORMAL,TOTIC,PRINC,PROB,
     *BOUND,WAGE,CYCLE,WACY,NTASK,NCDEC,NCOLUM,NDTOP,NSTAGE,NCSTA,IABORT,
     *ICEL,IKARA,IFLAG,STATUS,NTDEC,IDEC,MARK,NDEC,NPRE,IMPRE,ISTA,NSTA,
     *NTFOL,FOLLOW,ISDF,IDMT,NPREVS,NPREVD,IGDEC,IDUMY,IMFOL,NTSTA,
     *COLUMN,NFOL,NTPRE,PREC
C
      DO 90 I=1,LDD
      DO 90 J=1,30
   90 IGDEC(I,J) = 0
      I = 1
      N1 = NTSTA(NSTAGE)
      DO 100 M1=1,N1
      DO 110 M2=1,NTASK
      MARK(M2) = 0
  110 STATUS(M2) = 0
      MT1 = NCSTA + NTSTA(NSTAGE - 1) + M1
      MT2 = NSTA(MT1)
      DO 120 M3=1,MT2
  120 STATUS(ISTA(MT1,M3)) = 1
      NMARK = 0
      DO 130 M4=1,NTASK
      IF(STATUS(M4).NE.0) GO TO 130
      IF(NPRE(M4).EQ.0) GO TO 140
      MT3 = NPRE(M4)
      DO 150 M6=1,MT3
      IF(STATUS(IMPRE(M4,M6)).NE.1) GO TO 130
  150 CONTINUE
  140 NMARK = NMARK + 1
      MARK(NMARK) = M4
  130 CONTINUE
      IF(NMARK.EQ.0) GO TO 100
      CALL DECVAR(NMARK,I,MT1)
      IF(IABORT.GT.0) GO TO 999
      M88 = I + NDTOP - 1
      DO 160 IE=I,M88
  160 IGDEC(IE,1) = M1
      NTDEC(NSTAGE) = NTDEC(NSTAGE) + NDTOP
      IF(M1.EQ.1) THEN
      I = I + NDTOP
      ELSE
      CALL DECELM(I)
      ENDIF
      NTDEC(NSTAGE) = I - 1
  100 CONTINUE
      NTDEC(NSTAGE) = I - 1
  999 RETURN
      END
C
C**** SUBROUTINE "DECELM" ELIMINATES DECISION VARIABLES GENERATED
C**** IN SUBROUTINE "DCSION" IF THE DECISION VARIABLES HAVE BEEN
C**** GENERATED EARLIER
C
      SUBROUTINE DECELM(I)
      PARAMETER (LW=5,LTFP=20,LST=15000,LDE=15000,LDD=15000,LTASK=50)
      DIMENSION COSTS(LST),COST(LDE),VAR(LTASK),TOTIC(LTASK)
      REAL MEAN(LTASK),INCOM(LTASK),NORMAL(310),PRINC(LDE,LTASK)
      INTEGER*2 IGDEC(LDD,30),NTDEC(0:LTASK),NTSTA(LTASK),NSTA(LST),
     *ISTA(LST,LTASK),NDEC(LDE),IDEC(LDE,LTASK),NPREVS(LDE),
     *NPREVD(LST),IDUMY(LDD),MARK(LTASK),STATUS(LTASK),
     *NPRE(LTASK),IMPRE(LTASK,LTFP),COLUMN(LTASK),NFOL(LTASK),
     *IMFOL(LTASK,LTFP),NTPRE(LTASK),PREC(LTASK,LTASK),
     *NTFOL(LTASK),FOLLOW(LTASK,LTASK),IDMT(LTASK),ISDF(LTASK,LTASK)
   COMMON /MODEL/COST,COSTS,INCOM,MEAN,VAR,NORMAL,TOTIC,PRINC,PROB,
  *BOUND,WAGE,CYCLE,WACY,NTASK,NCDEC,NCOLUM,NDTOP,NSTAGE,NCSTA,IABORT,
  *ICEL,IKARA,IFLAG,STATUS,NTDEC,IDEC,MARK,NDEC,NPRE,IMPRE,ISTA,NSTA,
  *NTFOL,FOLLOW,ISDF,IDMT,NPREVS,NPREVD,IGDEC,IDUMY,IMFOL,NTSTA,
  *COLUMN,NFOL,NTPRE,PREC
C
   JA = NTDEC(NSTAGE)
   DO 100 JB=1,JA
100 IDUMY(JB) = 1
   JB = JA - 1
   DO 110 JC=1,JB
   IF(IDUMY(JC).EQ.0) GO TO 110
   JD = JC + 1
   DO 120 JE=JD,JA
   IF(IDUMY(JE).EQ.0) GO TO 120
   JF = NCDEC + NTDEC(NSTAGE - 1) + JC
   JG = NCDEC + NTDEC(NSTAGE - 1) + JE
   IF(NDEC(JF).NE.NDEC(JG)) GO TO 120
   IF(COST(JF).NE.COST(JG)) GO TO 120
   JH = NDEC(JF)
   JI = NDEC(JG)
   DO 130 JK=1,JH
   DO 140 JL=1,JI
   IF(IDEC(JF,JK).EQ.IDEC(JG,JL)) GO TO 130
140 CONTINUE
   GO TO 120
130 CONTINUE
   IDUMY(JE) = 0
   IA = 0
   DO 150 JK=1,30
   IF(IGDEC(JC,JK).EQ.0) GO TO 160
   IA = IA + 1
150 CONTINUE
160 DO 170 JL=1,30
   IF(IGDEC(JE,JL).EQ.0) GO TO 120
   DO 180 JM=1,IA
   IF(IGDEC(JE,JL).EQ.IGDEC(JC,JM)) GO TO 170
180 CONTINUE
   IGDEC(JC,IA+1) = IGDEC(JE,JL)
   IA = IA + 1
170 CONTINUE
120 CONTINUE
110 CONTINUE
   I = 1
   LINE = NCDEC + NTDEC(NSTAGE - 1) + 1
   DO 190 JB=1,JA
   IF(IDUMY(JB).EQ.0) GO TO 190
   JD = NCDEC + NTDEC(NSTAGE - 1) + JB
   MK2 = NDEC(JD)
   DO 200 JM=1,MK2
   PRINC(LINE,JM) = PRINC(JD,JM)
200 IDEC(LINE,JM) = IDEC(JD,JM)
   NDEC(LINE) = MK2
   COST(LINE) = COST(JD)
   DO 210 M17=1,30
   IGDEC(I,M17) = IGDEC(JB,M17)
210 CONTINUE
   I = I + 1
   LINE = LINE + 1
190 CONTINUE
   MK4 = NCDEC + NTDEC(NSTAGE - 1) + I
   MK5 = NCDEC + NTDEC(NSTAGE - 1) + NTDEC(NSTAGE)
   DO 220 M18=MK4,MK5
   DO 230 M15=1,10
   PRINC(M18,M15) = 0.0
230 IDEC(M18,M15) = 0
   COST(M18) = 0.0
220 CONTINUE
   DO 240 M19=I,JA
   DO 240 M20=1,30
   IGDEC(M19,M20) = 0
240 CONTINUE
```

```
      RETURN
      END
C
C**** SUBROUTINE 'REPORT' PRINTS THE SOLUTION OF THE PROBLEM
C
      SUBROUTINE REPORT(JEND)
      PARAMETER (LW=5,LTFP=20,LST=15000,LDE=15000,LDD=15000,LTASK=50)
      DIMENSION COSTS(LST),COST(LDE),VAR(LTASK),TOTIC(LTASK)
      REAL MEAN(LTASK),INCOM(LTASK),NORMAL(310),PRINC(LDE,LTASK)
      INTEGER*2 IGDEC(LDD,30),NTDEC(0:LTASK),NTSTA(LTASK),NSTA(LST),
     *ISTA(LST,LTASK),NDEC(LDE),IDEC(LDE,LTASK),NPREVS(LDE),
     *NPREVD(LST),IDUMY(LDD),MARK(LTASK),STATUS(LTASK),
     *NPRE(LTASK),IMPRE(LTASK,LTFP),COLUMN(LTASK),NFOL(LTASK),
     *IMFOL(LTASK,LTFP),NTPRE(LTASK),PREC(LTASK,LTASK),
     *NTFOL(LTASK),FOLLOW(LTASK,LTASK),IDMT(LTASK),ISDF(LTASK,LTASK)
      COMMON /MODEL/COST,COSTS,INCOM,MEAN,VAR,NORMAL,TOTIC,PRINC,PROB,
     *BOUND,WAGE,CYCLE,WACY,NTASK,NCDEC,NCOLUM,NDTOP,NSTAGE,NCSTA,IABORT,
     *ICEL,IKARA,IFLAG,STATUS,NTDEC,IDEC,MARK,NDEC,NPRE,IMPRE,ISTA,NSTA,
     *NTFOL,FOLLOW,ISDF,IDMT,NPREVS,NPREVD,IGDEC,IDUMY,IMFOL,NTSTA,
     *COLUMN,NFOL,NTPRE,PREC
C
      J = NTSTA(NSTAGE + 1)
      DO 100 K=1,J
      IA = NCSTA + NTSTA(NSTAGE) + K
      IF((NSTA(IA)).LT.NTASK) GO TO 100
      JEND = 1
      WRITE(LW,900) NSTAGE,COSTS(IA)
      WRITE(LW,905) NSTAGE,(IDEC(NPREVD(IA),IC),IC=1,NDEC(NPREVD(IA)))
      NEWSTA = NSTAGE
  110 IF(NEWSTA.LT.2) RETURN
      IB = NPREVS(IA)
      NEWSTA = NEWSTA - 1
      WRITE(LW,905) NEWSTA,(IDEC(NPREVD(IB),IC),IC=1,NDEC(NPREVD(IB)))
      IA = IB
      GO TO 110
  100 CONTINUE
      RETURN
  900 FORMAT(//,4X,'MIN DESIGN COST WITH ',I2,' STATIONS IS',F9.3,
     *' $/UNIT',/)
  905 FORMAT(10X,'TASKS OF STATION',I3,' ARE :',4X,10(I3,','))
      END
C
C**** FUNCTION "MATRIX" GENERATES COMBINATIONS OF ELEMENTS IN A SET
C
      FUNCTION MATRIX(I,J)
      MATRIX = 0
      IF(J.EQ.1) GO TO 10
      K = ( I / ( INT ( 2 ** ( J - 1 )))) + 1
      IF(MOD(K,2).EQ.1) RETURN
      MATRIX = 1
      RETURN
   10 IF(MOD(I,2).NE.1) RETURN
      MATRIX = 1
      RETURN
      END
C
C
      SUBROUTINE PRCA2(SANTAK,FNORM)
      PARAMETER (LW=5,LTFP=20,LST=15000,LDE=15000,LDD=15000,LTASK=50)
      DIMENSION COSTS(LST),COST(LDE),VAR(LTASK),TOTIC(LTASK)
      REAL MEAN(LTASK),INCOM(LTASK),NORMAL(310),PRINC(LDE,LTASK)
      INTEGER*2 IGDEC(LDD,30),NTDEC(0:LTASK),NTSTA(LTASK),NSTA(LST),
     *ISTA(LST,LTASK),NDEC(LDE),IDEC(LDE,LTASK),NPREVS(LDE),
     *NPREVD(LST),IDUMY(LDD),MARK(LTASK),STATUS(LTASK),
     *NPRE(LTASK),IMPRE(LTASK,LTFP),COLUMN(LTASK),NFOL(LTASK),
     *IMFOL(LTASK,LTFP),NTPRE(LTASK),PREC(LTASK,LTASK),
     *NTFOL(LTASK),FOLLOW(LTASK,LTASK),IDMT(LTASK),ISDF(LTASK,LTASK)
      COMMON /MODEL/COST,COSTS,INCOM,MEAN,VAR,NORMAL,TOTIC,PRINC,PROB,
     *BOUND,WAGE,CYCLE,WACY,NTASK,NCDEC,NCOLUM,NDTOP,NSTAGE,NCSTA,IABORT,
     *ICEL,IKARA,IFLAG,STATUS,NTDEC,IDEC,MARK,NDEC,NPRE,IMPRE,ISTA,NSTA,
     *NTFOL,FOLLOW,ISDF,IDMT,NPREVS,NPREVD,IGDEC,IDUMY,IMFOL,NTSTA,
     *COLUMN,NFOL,NTPRE,PREC
C
      IF(FNORM) 110,120,120
  110 IF(FNORM.LT.-3.0) GO TO 130
      J = (0.01 - FNORM) * 100.
      SANTAK = NORMAL(J)
      RETURN
  130 SANTAK = 1.
      RETURN
  120 IF(FNORM.GT.3.0) GO TO 140
      J = (FNORM + 0.01) * 100.
      SANTAK = 1. - NORMAL(J)
      RETURN
  140 SANTAK = 0.0
      RETURN
      END
C
C
      FUNCTION MATCOM(I,J)
      MATCOM = 0
      IF(I.EQ.1)  RETURN
      IF(J.EQ.1) GO TO 10
      K = ( (I-1) / ( INT ( 2 ** ( J - 1 )))) + 1
      IF(MOD(K,2).EQ.1) RETURN
      MATCOM = 1
      RETURN
   10 IF(MOD(I+1,2).NE.1) RETURN
      MATCOM = 1
      RETURN
      END
C
C
      SUBROUTINE PRCMB(MT1,K,SANCOM)
      PARAMETER (LW=5,LTFP=20,LST=15000,LDE=15000,LDD=15000,LTASK=50,
     *LDAL=5000)
      DIMENSION COSTS(LST),COST(LDE),VAR(LTASK),TOTIC(LTASK),
     *PRINC(LDE,LTASK)
      REAL MEAN(LTASK),INCOM(LTASK),NORMAL(310),PONCE(LDAL),
     *DMEAN(LDAL),DVAR(LDAL),DSANS(LDAL)
      INTEGER*2 IGDEC(LDD,30),NTDEC(0:LTASK),NTSTA(LTASK),NSTA(LST),
     *ISTA(LST,LTASK),NDEC(LDE),IDEC(LDE,LTASK),NPREVS(LDE),
     *NPREVD(LST),IDUMY(LDD),MARK(LTASK),STATUS(LTASK),
     *NPRE(LTASK),IMPRE(LTASK,LTFP),COLUMN(LTASK),NFOL(LTASK),
     *IMFOL(LTASK,LTFP),NTPRE(LTASK),PREC(LTASK,LTASK),
     *NTFOL(LTASK),FOLLOW(LTASK,LTASK),NST(LTASK),LABEL(LDAL),
     *FINISH(LTASK),NSTART(LTASK,LTASK),IS(LTASK,LTASK),NIS(LTASK),
     *INFO(LDAL,LTASK),DAL,BNO,FINO,IDMT(LTASK),ISDF(LTASK,LTASK)
      COMMON /MODEL/COST,COSTS,INCOM,MEAN,VAR,NORMAL,TOTIC,PRINC,PROB,
     *BOUND,WAGE,CYCLE,WACY,NTASK,NCDEC,NCOLUM,NDTOP,NSTAGE,NCSTA,IABORT,
     *ICEL,IKARA,IFLAG,STATUS,NTDEC,IDEC,MARK,NDEC,NPRE,IMPRE,ISTA,NSTA,
     *NTFOL,FOLLOW,ISDF,IDMT,NPREVS,NPREVD,IGDEC,IDUMY,IMFOL,NTSTA,
     *COLUMN,NFOL,NTPRE,PREC
      COMMON /EKLEME/BNO,FINO,NST,NSTART,FINISH
C
      KC = NPREVD(MT1)
      IZ = NPREVS(MT1)
      NSTM1 = NSTAGE - 1
      DO 100 I=1,NSTM1
      DO 120 J=1,NDEC(KC)
      IS(NSTM1-I+1,J) = IDEC(KC,J)
  120 CONTINUE
      NIS(NSTM1-I+1) = NDEC(KC)
      KC = NPREVD(IZ)
      IZ = NPREVS(IZ)
  100 CONTINUE
      DO 110 I=1,NTASK
      INFO(1,I) = 0
  110 CONTINUE
      DO 111 I=1,LDAL
      LABEL(I) = 0
  111 CONTINUE
      DAL = 1
   DSANS(1) = 1.0
   DMEAN(1) = 0.0
   DVAR(1) = 0.0
   IESKI = 1
   IYENI = 1
   PONCE(1) = 1.0
   DO 130 I=1,NSTM1
   DO 140 J=1,NIS(I)
   ISIJ = IS(I,J)
   DO 160 IA=IESKI,IYENI
   IF(LABEL(IA).EQ.1) GO TO 160
   IF(NTPRE(ISIJ).EQ.0) GO TO 162
   DO 165 IB=1,NTPRE(ISIJ)
   IF(INFO(IA,PREC(ISIJ,IB)).EQ.0) GO TO 200
165 CONTINUE
162 DAL = DAL + 1
   DO 170 IB=1,NTASK
170 INFO(DAL,IB) = INFO(IA,IB)
   IF(J.EQ.1) THEN
   PONCE(DAL) = DSANS(IA)
   DMEAN(DAL) = MEAN(ISIJ)
   DVAR(DAL) = VAR(ISIJ)
   ELSE
   PONCE(DAL) = PONCE(IA)
   DMEAN(DAL) = DMEAN(IA) + MEAN(ISIJ)
   DVAR(DAL) = DVAR(IA) + VAR(ISIJ)
   ENDIF
   INFO(DAL,ISIJ) = 1
   IF(DVAR(DAL).LE.0.0) DVAR(DAL) = 0.0001
   FNORM = (CYCLE - DMEAN(DAL)) / SQRT(DVAR(DAL))
   CALL PRCA2(DPROB, FNORM)
   DSANS(DAL) = (1. - DPROB) * PONCE(DAL)
   IF(DSANS(DAL).LE.0.0) LABEL(DAL) = 1
   DSANS(DAL + 1) = DSANS(IA) - DSANS(DAL)
   DAL = DAL + 1
   IF(DSANS(DAL).LE.0.0) LABEL(DAL) = 1
   GO TO 205
200 DAL = DAL + 1
   DSANS(DAL) = DSANS(IA)
205 DO 175 IB=1,NTASK
175 INFO(DAL,IB) = INFO(IA,IB)
   IF(J.EQ.1) THEN
   PONCE(DAL) = DSANS(IA)
   ELSE
   PONCE(DAL) = PONCE(IA)
   ENDIF
   DO 180 IB=1,FINO
   IF(FINISH(IB).EQ.ISIJ) LABEL(DAL) = 1
180 CONTINUE
160 CONTINUE
   IESKI = IYENI + 1
   IYENI = DAL
140 CONTINUE
130 CONTINUE
   SANCOM = 0.0
   DO 206 I=IESKI,IYENI
   IF(LABEL(I).EQ.1) GO TO 206
   IF(BNO.EQ.0) GO TO 260
   DO 210 J=1,BNO
   DO 250 IA=1,NST(J)
   IF(INFO(I,NSTART(J,IA)).EQ.0) GO TO 210
250 CONTINUE
   GO TO 206
210 CONTINUE
260 SANCOM = SANCOM + DSANS(I)
206 CONTINUE
   RETURN
   END
```

Appendix B. FORTRAN Code Of The Improvement Procedure

```
C
C    THIS PROGRAM APPLIES THE IMPROVEMENT PROCEDURE TO THE SINGLE-
C    MODEL, STOCHASTIC ASSEMBLY LINE BALANCING PROBLEM
C
C    MAXIMUM NUMBER OF TASKS < 100
C    MAXIMUM NUMBER OF NODES < 10000
C
C    VARIABLES OF THE PROGRAM:
C
C    CASSIG(I)  = TCN OF NODE I
C    COLUMN(I)  = NUMBER OF THE COLUMN AT WHICH TASK I IS LOCATED
C            ON THE PRECEDENCE DIAGRAM
C    COST(I)    = APPROXIMATE COST OF NODE I
C    CYCLE      = CYCLE TIME
C    ELEMAN(I,J) = IDENTITY OF THE JTH TASK IN NODE I
C    GSTR(I)    = 1 IF NODE I IS PRUNED, OTHERWISE 0
C    IABORT     = 1 IF THE PROGRAM IS TERMINATED, OTHERWISE 0
C    IC(I)      = CUMULATIVE INCOMPLETION COST OF TASK I
C    IMFOL(I,J)  = IDENTITY OF THE JTH IMMEDIATE FOLLOWER OF TASK I
C    IMPRE(I,J)  = IDENTITY OF THE JTH IMMEDIATE PREDECESSOR OF TASK I
C    INCOM(I)   = INCOMPLETION COST OF TASK I
C    KLSTA      = NUMBER OF STATIONS IN THE INITIAL SOLUTION
C    LEVEL      = CURRENT LEVEL OF THE TREE AT WHICH
C            NODES ARE BEING BRANCHED
C    MARK       = LIST OF TASKS AVAILABLE FOR ASSIGNMENT
C    MEAN(I)    = EXPECTED PERFORMANCE TIME OF TASK I
C    NCOLUM     = NUMBER OF COLUMNS ON THE PRECEDENCE DIAGRAM
C    NFOL(I)    = NUMBER OF IMMEDIATE FOLLOWERS OF TASK I
C    NLEVEL     = NUMBER OF NODES AT LEVEL "LEVEL"
C    NMARK      = NUMBER OF TASKS IN LIST "MARK"
C    NODE       = COUNTER FOR THE NUMBER OF NODES GENERATED
C    NORMAL     = LIST WHICH CONTAINS NORMAL DISTRIBUTION VALUES
C    NPRE(I)    = NUMBER OF IMMEDIATE PREDECESSORS OF TASK I
C    NTASK      = NUMBER OF TASKS IN THE PROBLEM
C    NTNODE(I)  = NUMBER OF TASKS IN NODE I
C    PARENT(I)  = PARENT NODE OF NODE I
C    TIME       = SUBROUTINE THAT TURNS ON THE TIMER TO DETERMINE
C            THE AMOUNT OF CPU TIME USED
C    TIMECK     = SUBROUTINE THAT DETERMINES THE AMOUNT OF CPU TIME
```

```
C             USED SINCE THE LAST CALL TO SUBROUTINE "TIMEON"
C    UB          = CURRENT UPPER BOUND
C    VAR(I)      = VARIANCE OF THE PERFORMANCE TIME OF TASK I
C    VMULT       = VARIANCE MULTIPLIER
C    WACY        = LABOR COST OF A STATION
C    WAGE        = LABOR RATE
C
C
C
C**** MAIN PROGRAM
C
      PARAMETER (LN = 10000,LT = 100,LTD = 100)
      DIMENSION VAR(LT),COST(LN),CASSIG(LN),TMEAN(LT)
      REAL MEAN(LT),IC(LT),NORMAL(310),PRINC(LN,LT),INCOM(LT)
      INTEGER*2 NPRE(LT),MARK(LT),ELEMAN(LN,LT),NTNODE(LN),PARENT(LN),
     *IMPRE(LT,10),NFOL(LT),NTPRE(LT),PREC(LT,LT),
     *IMFOL(LT,10),STATUS(LT),LIST(LT),MJOB(LT),MACH(LT,LT),GSTR(LN),
     *NTFOL(LT),FOLLOW(LT,LT),IDMT(LTD),ISDF(LTD,LTD)
      COMMON INCOM,PRINC,VAR,COST,CASSIG,MEAN,IC,NORMAL,TMEAN,WACY,UB,
     *VMULT,CMULT,RHOPL1,CYCLE,ALT,UST,NTASK,LEVEL,KLSTA,NGBR,ICEL,
     *IKARA,IFLAG,IABORT,NODE,NLEVEL,NLM2,IA,MARK,NPRE,NTPRE,PREC,NTFOL,
     *FOLLOW,ISDF,IDMT,ELEMAN,GSTR,NTNODE,PARENT,IMPRE,NFOL,IMFOL,
     *STATUS,LIST,MJOB,MACH
C
      CALL TIMEON
      CALL INIT
      CALL TIMECK(NTIME)
      TIME = NTIME / 100.
      WRITE(6,900) TIME
      DO 10 LEVEL= 1,NTASK
      IF(LEVEL.GT.1)  GO TO 12
      NMARK = 0
      DO 15 I = 1,NTASK
      IF(NPRE(I).GT.0)  GO TO 15
      NMARK = NMARK + 1
      MARK(NMARK) = I
   15 CONTINUE
      CALL GNRTOR(NMARK,0)
      IF(IABORT.GT.0)  GO TO 999
      NLM2 = 0
      NLEVEL = NODE - 1
      GO TO 14
   12 CALL NODGEN
      CALL TIMECK(NTIME)
      TIME = NTIME / 100.
      WRITE(6,905) LEVEL,TIME
   14 IF(IABORT.GT.0)  GO TO 999
      IF(TIME.GT.120.)  GO TO 950
   10 CONTINUE
  950 WRITE(6,920)
  999 WRITE(6,925) NODE
      STOP
C
  900 FORMAT(/,5X,'CPU TIME SPENT FOR READING DATA AND INITIALIZATION ='
     *,F8.3,' SECONDS')
  905 FORMAT(/,5X,'CPU TIME SPENT AT THE END OF',I4,'TH LEVEL =',
     *F8.3,' SECONDS')
  925 FORMAT(/,5X,'TOTAL NUMBER OF NODES GENERATED =',I5)
  920 FORMAT(5X,'CPU TIME LIMIT IS EXCEEDED')
      END
C
C**** SUBROUTINE "INIT" READS DATA, INITIALIZES VARIABLES
C
      SUBROUTINE INIT
      PARAMETER (LN = 10000,LT = 100,LTD = 100)
      DIMENSION VAR(LT),COST(LN),CASSIG(LN),TMEAN(LT)
      REAL MEAN(LT),IC(LT),NORMAL(310),PRINC(LN,LT),INCOM(LT)
      INTEGER*2 NPRE(LT),MARK(LT),ELEMAN(LN,LT),NTNODE(LN),PARENT(LN),
     *IMPRE(LT,10),NFOL(LT),NTPRE(LT),PREC(LT,LT),IMFOL(LT,10),
     *STATUS(LT),LIST(LT),COLUMN(LT),MJOB(LT),MACH(LT,LT),
     *GSTR(LN),NTFOL(LT),FOLLOW(LT,LT),IDMT(LTD),ISDF(LTD,LTD)
      COMMON INCOM,PRINC,VAR,COST,CASSIG,MEAN,IC,NORMAL,TMEAN,WACY,UB,
     *VMULT,CMULT,RHOPL1,CYCLE,ALT,UST,NTASK,LEVEL,KLSTA,NGBR,ICEL,
     *IKARA,IFLAG,IABORT,NODE,NLEVEL,NLM2,IA,MARK,NPRE,NTPRE,PREC,NTFOL,
     *FOLLOW,ISDF,IDMT,ELEMAN,GSTR,NTNODE,PARENT,IMPRE,NFOL,IMFOL,
     *STATUS,LIST,MJOB,MACH
C
      READ(4,*,END=990) IFLAG
      IF(IFLAG.EQ.0)  GO TO 5
      READ(4,*,END=990) ICEL,WAGE,CYCLE,IKARA,VMULT,CMULT,UB,RHO,
     *KLSTA,NGBR
      DO 10 I=1,ICEL
      READ(4,*,END=990) MEAN(I),COLUMN(I),NFOL(I),NPRE(I)
      IF(NFOL(I).EQ.0)  GO TO 12
      READ(4,*,END=990) (IMFOL(I,J),J=1,NFOL(I))
 12   IF(NPRE(I).EQ.0)  GO TO 10
      READ(4,*,END=990) (IMPRE(I,J),J=1,NPRE(I))
 10   CONTINUE
      DO 100 I=1,ICEL
      DO 120 J=1,ICEL
  120 STATUS(J)=0
      STATUS(I) = 1
      K=COLUMN(I)
      DO 150 N=K,IKARA
      DO 200 J=1,ICEL
      IF(COLUMN(J).NE.N)  GO TO 200
      IF(STATUS(J).NE.1)  GO TO 200
      IF(NFOL(J).EQ.0)    GO TO 200
      IAA = NFOL(J)
      DO 250 IB=1,IAA
  250 STATUS(IMFOL(J,IB)) = 1
  200 CONTINUE
  150 CONTINUE
      TOTIC = 0.0
      DO 300 J=1,ICEL
      IF(STATUS(J).LT.1)  GO TO 300
      TOTIC = TOTIC + MEAN(J)
  300 CONTINUE
      IC(I) = TOTIC * CMULT
      INCOM(I) = MEAN(I) * CMULT
  100 CONTINUE
      DO 189 I=1,ICEL
  189 IDMT(I) = 0
      DO 600 I=1,ICEL
      K = COLUMN(I)
      IF(K.EQ.IKARA)  GO TO 600
      IF(NFOL(I).EQ.0) GO TO 600
      DO 610 J=I,ICEL
  610 STATUS(J) = 0
      DO 620 J=1,NFOL(I)
  620 STATUS(IMFOL(I,J)) = 1
      DO 630 N=K,IKARA
      DO 640 J=I,ICEL
      IF(COLUMN(J).NE.N)   GO TO 640
      IF(STATUS(J).EQ.0)   GO TO 640
      IF(NFOL(J).EQ.0)     GO TO 640
      DO 650 IB=1,NFOL(J)
  650 STATUS(IMFOL(J,IB)) = 1
  640 CONTINUE
  630 CONTINUE
      DO 660 J=I,ICEL
      IF(STATUS(J).LT.1)  GO TO 660
      IDMT(I) = IDMT(I) + 1
      ISDF(I,IDMT(I)) = J
  660 CONTINUE
  600 CONTINUE
    5 READ(4,*,END=990) NTASK,WAGE,CYCLE,NCOLUM,VMULT,CMULT,UB,RHO,
     *KLSTA,NGBR
      DO 1110 I=1,NTASK
      READ(4,*,END=990) MEAN(I),COLUMN(I),NFOL(I),NPRE(I)
      IF(NFOL(I).EQ.0)  GO TO 1112
      READ(4,*,END=990) (IMFOL(I,J),J=1,NFOL(I))
 1112 IF(NPRE(I).EQ.0)  GO TO 1110
      READ(4,*,END=990) (IMPRE(I,J),J=1,NPRE(I))
```

```
1110 CONTINUE
   READ(3,*,END=990) (NORMAL(I),I=1,310)
   IF(IFLAG.EQ.1) GO TO 7000
   DO 109 I=1,NTASK
   DO 122 J=1,NTASK
 122 STATUS(J)=0
   STATUS(I) = 1
   K=COLUMN(I)
   DO 151 N=K,NCOLUM
   DO 201 J=1,NTASK
   IF(COLUMN(J).NE.N) GO TO 201
   IF(STATUS(J).NE.1) GO TO 201
   IF(NFOL(J).EQ.0)  GO TO 201
   IAA = NFOL(J)
   DO 251 IB=1,IAA
 251 STATUS(IMFOL(J,IB)) = 1
 201 CONTINUE
 151 CONTINUE
   TOTIC = 0.0
   DO 301 J=1,NTASK
   IF(STATUS(J).LT.1) GO TO 301
   TOTIC = TOTIC + MEAN(J)
 301 CONTINUE
   IC(I) = TOTIC * CMULT
   INCOM(I) = MEAN(I) * CMULT
 109 CONTINUE
7000 DO 181 I=1,NTASK
   NTFOL(I) = 0
 181 NTPRE(I) = 0
   DO 191 I=1,NTASK
   K = COLUMN(I) - 1
   IF(K.EQ.0) GO TO 191
   DO 182 J=1,I
 182 STATUS(J) = 0
   DO 183 J=1,NPRE(I)
 183 STATUS(IMPRE(I,J)) = 1
   DO 176 N=1,K
   M = K - N + 1
   DO 171 J=1,I
   IF(COLUMN(J).NE.M) GO TO 171
   IF(STATUS(J).EQ.0) GO TO 171
   IF(NPRE(J).EQ.0)  GO TO 171
   DO 164 IB=1,NPRE(J)
 164 STATUS(IMPRE(J,IB)) = 1
 171 CONTINUE
 176 CONTINUE
   DO 1160 J=1,I
   IF(STATUS(J).LT.1) GO TO 1160
   NTPRE(I) = NTPRE(I) + 1
   PREC(I,NTPRE(I)) = J
1160 CONTINUE
 191 CONTINUE
   DO 1600 I=1,NTASK
   K = COLUMN(I)
   IF(K.EQ.NCOLUM) GO TO 1600
   IF(NFOL(I).EQ.0) GO TO 1600
   DO 1610 J=1,NTASK
1610 STATUS(J) = 0
   DO 1620 J=1,NFOL(I)
1620 STATUS(IMFOL(I,J)) = 1
   DO 1630 N=K,NCOLUM
   DO 1640 J=1,NTASK
   IF(COLUMN(J).NE.N)  GO TO 1640
   IF(STATUS(J).EQ.0)  GO TO 1640
   IF(NFOL(J).EQ.0)   GO TO 1640
   DO 1650 IB=1,NFOL(J)
1650 STATUS(IMFOL(J,IB)) = 1
1640 CONTINUE
1630 CONTINUE
   DO 1660 J=1,NTASK
   IF(STATUS(J).LT.1) GO TO 1660
   NTFOL(I) = NTFOL(I) + 1
```

```
      FOLLOW(I,NTFOL(I)) = J
 1660 CONTINUE
 1600 CONTINUE
      NODE = 1
      WACY = WAGE * CYCLE / 60.
      RHOPL1 = RHO + 1.
      DO 99 I=1,NTASK
      DUMMY = MEAN(I) * VMULT
   99 VAR(I) = DUMMY * DUMMY
      DO 1 I=1,NTASK
    1 STATUS(I) = 0
      DO 2 I=1,LN
      CASSIG(I) = 0.0
    2 GSTR(I) = 0
      DUMMY = 0.5 * SQRT(VMULT * (49. * CYCLE + 150.06 * VMULT ))
      ALT = CYCLE + 6.13 * VMULT - DUMMY
      UST = ALT + 2. * DUMMY
C
      WRITE(6,900) CYCLE,WAGE,UB,RHO,KLSTA,NGBR,VMULT,CMULT
      WRITE(6,905)
      DO 20 I=1,NTASK
      CDUM = MEAN(I) * CMULT
      IF(NFOL(I).EQ.0)  GO TO 22
      WRITE(6,910) I,MEAN(I),VAR(I),CDUM,IC(I),(IMFOL(I,J),J=1,NFOL(I))
      GO TO 20
   22 WRITE(6,915) I,MEAN(I),VAR(I),CDUM,IC(I)
   20 CONTINUE
      IABORT = 0
      RETURN
  990 IABORT = 1
      WRITE(6,991)
      RETURN
C
  900 FORMAT(/,9X,'CYCLE TIME   =',F5.1,2X,'MINUTES',/,9X,'LABOR RATE   ='
     *,F5.2,2X,'$/HOUR',/,9X,'INITIAL UPPER BOUND =  ',F8.3,
     */,9X,'RHO VALUE PROVIDED   =  ',F8.3,//,9X,
     *'KOTTAS - LAO PROC. SOLN. # OF STATIONS  =',I4,/,9X,
     *'NEIGHBORHOOD VALUE FOR APP. SOLUTIONS   =',I4,//,9X,
     *'STANDARD DEVIATION MULTIPLIER    =',F8.3,/,9X,
     *'INCOMPLETION COST MULTIPLIER     =',F8.3,/)
  905 FORMAT(//,28X,'INCOMPLETION   CUMULATIVE    IMMEDIATE',/,4X,'TASK'
     *,3X,'MEAN',3X,'VARIANCE',4X,'COST',13X,'COST',6X,'FOLLOWERS',/)
  910 FORMAT(5X,I2,3X,F5.2,4X,F6.3,5X,F6.3,7X,F7.3,7X,10(I2,','))
  915 FORMAT(5X,I2,3X,F5.2,4X,F6.3,5X,F6.3,7X,F7.3,7X,'NONE')
  991 FORMAT(5X,'ERROR IN INPUT DATA FILES, LESS DATA THAN EXPECTED')
      END
C
C**** SUBROUTINE "NODGEN" SELECTS THE NODE TO BE BRANCHED
C
      SUBROUTINE NODGEN
      PARAMETER (LN=10000,LT=100,LTD=100)
      DIMENSION VAR(LT),COST(LN),CASSIG(LN),TMEAN(LT)
      REAL MEAN(LT),IC(LT),NORMAL(310),PRINC(LN,LT),INCOM(LT)
      INTEGER*2 NPRE(LT),MARK(LT),ELEMAN(LN,LT),NTNODE(LN),PARENT(LN),
     *IMPRE(LT,10),NFOL(LT),NTPRE(LT),PREC(LT,LT),
     *IMFOL(LT,10),STATUS(LT),LIST(LT),MJOB(LT),MACH(LT,LT),GSTR(LN),
     *NTFOL(LT),FOLLOW(LT,LT),IDMT(LTD),ISDF(LTD,LTD)
      COMMON INCOM,PRINC,VAR,COST,CASSIG,MEAN,IC,NORMAL,TMEAN,WACY,UB,
     *VMULT,CMULT,RHOPL1,CYCLE,ALT,UST,NTASK,LEVEL,KLSTA,NGBR,ICEL,
     *IKARA,IFLAG,IABORT,NODE,NLEVEL,NLM2,IA,MARK,NPRE,NTPRE,PREC,NTFOL,
     *FOLLOW,ISDF,IDMT,ELEMAN,GSTR,NTNODE,PARENT,IMPRE,NFOL,IMFOL,
     *STATUS,LIST,MJOB,MACH
C
      LMINU1 = LEVEL - 1
      IF(NLEVEL.GT.0)  GO TO 5
      WRITE(6,900)
      IABORT = 1
      RETURN
    5 IA = NLM2 + 1
      IB = NLM2 + NLEVEL
      NLM2 = NLM2 + NLEVEL
      NLEVEL = 0
    4 CTEST = 1000.
      DO 1 I=IA,IB
      IF(GSTR(I).GT.0)  GO TO 1
      IF(COST(I).GE.CTEST) GO TO 1
      ND = I
      CTEST = COST(I)
    1 CONTINUE
      GSTR(ND) = 1
      IF(CTEST.EQ.1000.)  RETURN
C
      NMARK = 0
      DO 15 I=1,NTASK
   15 STATUS(I) = 0
      DO 20 I=1,LMINU1
      IF(I.GT.1)  GO TO 21
      IE = NTNODE(ND)
      DO 25 J=1,IE
   25 STATUS(ELEMAN(ND,J)) = 1
      M = PARENT(ND)
      GO TO 20
   21 IE = NTNODE(M)
      DO 30 J=1,IE
   30 STATUS(ELEMAN(M,J)) = 1
      M = PARENT(M)
   20 CONTINUE
      DO 35 I=1,NTASK
      IF(STATUS(I).GT.0)  GO TO 35
      IF(NPRE(I).EQ.0)   GO TO 36
      DO 40 J=1,NPRE(I)
      IF(STATUS(IMPRE(I,J)).EQ.0)  GO TO 35
   40 CONTINUE
   36 NMARK =NMARK + 1
      MARK(NMARK) = I
   35 CONTINUE
      IF(NMARK.EQ.0)  GO TO 4
      NOLD = NODE
      CALL GNRTOR(NMARK,ND)
      IF(IABORT.GT.0)  RETURN
      NLEVEL = NLEVEL + NODE - NOLD
      GO TO 4
  900 FORMAT(//,8X,'PROGRAM TERMINATED, THERE ARE NO ACTIVE NODES LEFT')
      END
C
C**** SUBROUTINE "GNRTOR" GENERATES NODES FROM PARENT NODES
C
      SUBROUTINE GNRTOR(NMARK,ND)
      PARAMETER (LN=10000,LT=100,LTD=100,LNM5=LN-5)
      DIMENSION VAR(LT),COST(LN),CASSIG(LN),TMEAN(LT)
      REAL MEAN(LT),IC(LT),NORMAL(310),PRINC(LN,LT),INCOM(LT)
      INTEGER*2 NPRE(LT),MARK(LT),ELEMAN(LN,LT),NTNODE(LN),PARENT(LN),
     *IMPRE(LT,10),NFOL(LT),NTPRE(LT),PREC(LT,LT),
     *IMFOL(LT,10),STATUS(LT),LIST(LT),NTFOL(LT),FOLLOW(LT,LT),
     *NEL(15,525),NS(0:15),IFS(15,525,10),NIFS(15,525),MA(LT),
     *JA(LT),MJOB(LT),MACH(LT,LT),GSTR(LN),IDMT(LTD),ISDF(LTD,LTD)
      COMMON INCOM,PRINC,VAR,COST,CASSIG,MEAN,IC,NORMAL,TMEAN,WACY,UB,
     *VMULT,CMULT,RHOPL1,CYCLE,ALT,UST,NTASK,LEVEL,KLSTA,NGBR,ICEL,
     *IKARA,IFLAG,IABORT,NODE,NLEVEL,NLM2,IA,MARK,NPRE,NTPRE,PREC,NTFOL,
     *FOLLOW,ISDF,IDMT,ELEMAN,GSTR,NTNODE,PARENT,IMPRE,NFOL,IMFOL,
     *STATUS,LIST,MJOB,MACH
C
      INDF = 2 ** NMARK - 1
      LINE = 0
      KSTA = 1
      NS(0) = 0
      DO 8 J=1,NTASK
      JA(J) = 0
      IF(STATUS(J).EQ.1) THEN
      MA(J) = 1
      ELSE
      MA(J) = 0
      ENDIF
    8 CONTINUE
```

```
      DO 9 J=1,NMARK
    9 MA(MARK(J)) = 1
      DO 10 L=1,INDF
      M = 0
      DO 15 J=1,NMARK
      IF(MATRIX(L,J).EQ.0)  GO TO 15
      M = M + 1
      ELEMAN(NODE,M) = MARK(J)
   15 CONTINUE
      NTNODE(NODE) = M
      PARENT(NODE) = ND
      CALL DIZI
      IF(CASSIG(NODE).GE.UB)  GO TO 140
      CALL ALTSNR
      IF(IABORT.GT.0)  RETURN
      IF(COST(NODE).GT.UB)  GO TO 140
      LINE = LINE + 1
      NEL(KSTA,LINE) = M
      NODE = NODE + 1
      IF(NODE.GT.LNM5)  GO TO 900
      GO TO 10
  140 NEL(KSTA,LINE) = 0
   10 CONTINUE
  150 NS(KSTA) = LINE
      LINF = NODE - NS(KSTA) - 1
      DO 160 L=1,NS(KSTA)
      K = 0
      LINF = LINF + 1
      DO 170 J=1,NTASK
      IAA = 0
      IF(MA(J).EQ.1)  GO TO 170
      IF(STATUS(J).EQ.1)  GO TO 170
      DO 180 J1=1,NPRE(J)
      IF(STATUS(IMPRE(J,J1)).EQ.1)  GO TO 172
      DO 200 J2=1,NEL(KSTA,L)
      IF(IMPRE(J,J1).EQ.ELEMAN(LINF,J2))  GO TO 172
  200 CONTINUE
      GO TO 180
  172 IAA = IAA + 1
  180 CONTINUE
      IF(IAA.NE.NPRE(J))  GO TO 170
      K = K + 1
      IFS(KSTA,L,K) = J
      JA(J) = 1
  170 CONTINUE
      NIFS(KSTA,L) = K
  160 CONTINUE
      DO 210 L=1,NTASK
      IF(JA(L).EQ.1)  MA(L) = 1
  210 CONTINUE
      NDUM = 0
      LINE = 0
      LINF = NODE - NS(KSTA) - 1
      IB = NS(KSTA)
      DO 220 L=1,IB
      LINF = LINF + 1
      IF(NIFS(KSTA,L).EQ.0)  GO TO 220
      NDUM = NDUM + 1
      IKF = 2 ** NIFS(KSTA,L) - 1
      DO 230 IG=1,IKF
      ID = NEL(KSTA,L)
      DO 240 L1=1,ID
  240 ELEMAN(NODE,L1) = ELEMAN(LINF,L1)
      M = NEL(KSTA,L)
      IE = NIFS(KSTA,L)
      DO 250 J=1,IE
      IF(MATRIX(IG,J).EQ.0)  GO TO 250
      M = M + 1
      ELEMAN(NODE,M) = IFS(KSTA,L,J)
  250 CONTINUE
      NTNODE(NODE) = M
      PARENT(NODE) = ND
      CALL DIZI
      IF(CASSIG(NODE).GE.UB)  GO TO 260
      CALL ALTSNR
      IF(IABORT.GT.0)  RETURN
      IF(COST(NODE).GT.UB)  GO TO 260
      LINE = LINE + 1
      NEL(KSTA+1,LINE) = M
      NODE = NODE + 1
      IF(NODE.GT.LNM5)  GO TO 900
      GO TO 230
  260 NEL(KSTA+1,LINE) = 0
  230 CONTINUE
  220 CONTINUE
      IF(NDUM.EQ.0)  RETURN
      KSTA = KSTA + 1
      GO TO 150
  900 IABORT = 1
      WRITE(6,910)
  910 FORMAT(3X,'NUMBER OF NODES GENERATED IS TOO HIGH')
      RETURN
      END
C
C**** SUBROUTINE "DIZI" SEQUENCES TASKS IN NODES
C
      SUBROUTINE DIZI
      PARAMETER (LN=10000,LT=100,LTD=100)
      DIMENSION VAR(LT),COST(LN),CASSIG(LN),TMEAN(LT)
      REAL MEAN(LT),IC(LT),NORMAL(310),NEW,PRINC(LN,LT),INCOM(LT)
      INTEGER*2 NPRE(LT),MARK(LT),ELEMAN(LN,LT),NTNODE(LN),PARENT(LN),
     *IMPRE(LT,10),NFOL(LT),NTPRE(LT),PREC(LT,LT),
     *IMFOL(LT,10),STATUS(LT),LIST(LT),MJOB(LT),MACH(LT,LT),GSTR(LN),
     *NTFOL(LT),FOLLOW(LT,LT),IYEN(LT),LABEL(LT),IDMT(LTD),ISDF(LTD,LTD)
      COMMON INCOM,PRINC,VAR,COST,CASSIG,MEAN,IC,NORMAL,TMEAN,WACY,UB,
     *VMULT,CMULT,RHOPL1,CYCLE,ALT,UST,NTASK,LEVEL,KLSTA,NGBR,ICEL,
     *IKARA,IFLAG,IABORT,NODE,NLEVEL,NLM2,IA,MARK,NPRE,NTPRE,PREC,NTFOL,
     *FOLLOW,ISDF,IDMT,ELEMAN,GSTR,NTNODE,PARENT,IMPRE,NFOL,IMFOL,
     *STATUS,LIST,MJOB,MACH
C
      NJOB = NTNODE(NODE)
      LMNS1 = LEVEL - 1
      NJOBM1 = NJOB - 1
      IF(NJOB.GE.2)  GO TO 777
      T = MEAN(ELEMAN(NODE,1))
      CALL PROB(T,-1.,PR1,PR2,CYCLE)
      PRINC(NODE,1) = PR1
      GO TO 310
  777 DO 201 J=1,NJOBM1
      JPLUS1 = J + 1
      DO 251 I=JPLUS1,NJOB
      IF(IC(ELEMAN(NODE,J)).GE.IC(ELEMAN(NODE,I)))  GO TO 251
      NON = NPRE(ELEMAN(NODE,I))
      DO 241 K=1,NON
      IF(IMPRE(ELEMAN(NODE,I),K).EQ.ELEMAN(NODE,J)) GO TO 251
  241 CONTINUE
      LTEMP = ELEMAN(NODE,J)
      ELEMAN(NODE,J) = ELEMAN(NODE,I)
      ELEMAN(NODE,I) = LTEMP
  251 CONTINUE
  201 CONTINUE
      J = 1
      TMEAN(1) = MEAN(ELEMAN(NODE,1))
  100 DO 300 I=J,NJOBM1
      IPLUS1 = I + 1
      IMNS1 = I - 1
      TMEAN(IPLUS1) = TMEAN(I) + MEAN(ELEMAN(NODE,IPLUS1))
      IF(TMEAN(IPLUS1).GE.ALT)  GO TO 207
      PRINC(NODE,I) = 0.0
      PRINC(NODE,IPLUS1) = 0.0
      GO TO 300
  207 IF(TMEAN(I).LE.UST)  GO TO 208
      PRINC(NODE,I) = 1.0
      PRINC(NODE,IPLUS1) = 1.0
      GO TO 300
  208 IENT1 = ELEMAN(NODE,IPLUS1)
      IENT = ELEMAN(NODE,I)
      CALL PROB(TMEAN(I),TMEAN(IPLUS1),PR1,PR2,CYCLE)
      IF(I.EQ.1) THEN
      PRINC(NODE,I) = PR1
      ELSE
      PRINC(NODE,I) = PR1 - PRINC(NODE,IMNS1)
      ENDIF
      PRINC(NODE,IPLUS1) = PR2 - PR1
      BGEC = PRINC(NODE,I)
      BGEC1 = PRINC(NODE,IPLUS1)
      IF((MEAN(IENT1).GT.MEAN(IENT)).
     *AND.(IC(IENT1).LT.IC(IENT))) GO TO 300
      DO 305 K=1,NTPRE(IENT1)
      IF(PREC(IENT1,K).EQ.IENT) GO TO 300
  305 CONTINUE
      CUR = IC(IENT) * PRINC(NODE,I) + IC(IENT1) * PRINC(NODE,IPLUS1)
      TMEANA = TMEAN(I)
      TMEAN(I) = TMEAN(I) - MEAN(IENT) + MEAN(IENT1)
      CALL PROB(TMEAN(I),TMEAN(IPLUS1),PR1,PR2,CYCLE)
      IF(I.EQ.1) THEN
      PRINC(NODE,I) = PR1
      ELSE
      PRINC(NODE,I) = PR1 - PRINC(NODE,IMNS1)
      ENDIF
      PRINC(NODE,IPLUS1) = PR2 - PR1
      NEW = IC(IENT1) * PRINC(NODE,I) + IC(IENT) * PRINC(NODE,IPLUS1)
      IF(CUR.GT.NEW) GO TO 315
      PRINC(NODE,I) = BGEC
      PRINC(NODE,IPLUS1) = BGEC1
      TMEAN(I) = TMEANA
  300 CONTINUE
  310 DO 301 IT=1,NJOB
  301 PRINC(NODE,IT) = 0.0
      CALL COSCAL(FIAT)
      CASSIG(NODE) = CASSIG(PARENT(NODE)) + FIAT + WACY
      RETURN
  315 LTEMP = ELEMAN(NODE,IPLUS1)
      ELEMAN(NODE,IPLUS1) = ELEMAN(NODE,I)
      ELEMAN(NODE,I) = LTEMP
      J = I - 1
      GO TO 100
      END
C
C**** SUBROUTINE "ALTSNR" SOLVES THE RELAXED PROBLEMS
C**** CORRESPONDING TO THE NODES
C
      SUBROUTINE ALTSNR
      PARAMETER (LN=10000,LT=100,LTD=100)
      DIMENSION VAR(LT),COST(LN),CASSIG(LN),TMEAN(LT)
      REAL MEAN(LT),IC(LT),NORMAL(310),NEW,PRINC(LN,LT),INCOM(LT)
      INTEGER*2 NPRE(LT),MARK(LT),ELEMAN(LN,LT),NTNODE(LN),PARENT(LN),
     *IMPRE(LT,10),NFOL(LT),NTPRE(LT),PREC(LT,LT),
     *IMFOL(LT,10),STATUS(LT),LIST(LT),MJOB(LT),MACH(LT,LT),GSTR(LN),
     *NTFOL(LT),FOLLOW(LT,LT),IDMT(LTD),ISDF(LTD,LTD)
      COMMON INCOM,PRINC,VAR,COST,CASSIG,MEAN,IC,NORMAL,TMEAN,WACY,UB,
     *VMULT,CMULT,RHOPL1,CYCLE,ALT,UST,NTASK,LEVEL,KLSTA,NGBR,ICEL,
     *IKARA,IFLAG,IABORT,NODE,NLEVEL,NLM2,IA,MARK,NPRE,NTPRE,PREC,NTFOL,
     *FOLLOW,ISDF,IDMT,ELEMAN,GSTR,NTNODE,PARENT,IMPRE,NFOL,IMFOL,
     *STATUS,LIST,MJOB,MACH
C
      CALL TIMECK(NTIME)
      IF(NTIME.GT.12000) GO TO 900
      DEGER = 10000.
      NJOB = 0
      IKS = 0
      DO 5 I=1,NTASK
      IF(STATUS(I).EQ.1) GO TO 5
      DO 6 J=1,NTNODE(NODE)
      IF(ELEMAN(NODE,J).EQ.I) GO TO 5
    6 CONTINUE
```

```
      NJOB = NJOB + 1
      LIST(NJOB) = I
    5 CONTINUE
      IF(NJOB.GT.0) GO TO 8
      COST(NODE) = CASSIG(NODE)
      GSTR(NODE) = 1
      GO TO 800
    8 NJOBM1 = NJOB - 1
      KLALT = KLSTA - LEVEL - NGBR
      KLUST = KLSTA - LEVEL + NGBR
      IF(KLALT.LT.1) KLALT = 1
      IF(KLUST.GT.NJOB) KLUST = NJOB
      DO 1 NMACH=KLALT,KLUST
      DO 10 J=1,NJOB
   10 TMEAN(J) = 0.0
      CCYCLE = NMACH * CYCLE
      DUMMY = 0.5 * SQRT(VMULT * (49. * CCYCLE + 150.06 * VMULT))
      ALT1 = CCYCLE + 6.13 * VMULT - DUMMY
      UST1 = ALT1 + 2. * DUMMY
      DO 200 J=1,NJOBM1
      JPLUS1 = J + 1
      DO 250 I=JPLUS1,NJOB
      IF(IC(LIST(J)).GE.IC(LIST(I))) GO TO 250
      LTEMP = LIST(J)
      LIST(J) = LIST(I)
      LIST(I) = LTEMP
  250 CONTINUE
  200 CONTINUE
      J = 1
  290 TMEAN(1) = MEAN(LIST(1))
      DO 300 I=J,NJOBM1
      IPLUS1 = I + 1
      TMEAN(IPLUS1) = TMEAN(I) + MEAN(LIST(IPLUS1))
      IF(TMEAN(IPLUS1).LE.ALT1) GO TO 300
      IF(TMEAN(I).GE.UST1)      GO TO 300
      LI1 = LIST(IPLUS1)
      LI = LIST(I)
      TOPD = 0.0
      DO 292 MT=I,NJOB
      TOPD = TOPD + INCOM(LIST(MT))
  292 CONTINUE
      TOPD1 = TOPD - INCOM(LI)
      IF((MEAN(LI1).GE.MEAN(LI)).AND.(TOPD1.LE.TOPD)) GO TO 300
      CALL PROB(TMEAN(I),TMEAN(IPLUS1),PR1,PR2,CCYCLE)
      CUR = TOPD * PR1 + TOPD1 * PR2
      TMEANA = TMEAN(I)
      TMEAN(I) = TMEAN(I) - MEAN(LI) + MEAN(LI1)
      CALL PROB(TMEAN(I),TMEAN(IPLUS1),PR1,PR2,CCYCLE)
      TOPD1 = TOPD - INCOM(LIST(IPLUS1))
      NEW = TOPD * PR1 + TOPD1 * PR2
      IF(CUR.GT.NEW) GO TO 310
      TMEAN(I) = TMEANA
  300 CONTINUE
      GO TO 320
  310 LTEMP = LIST(IPLUS1)
      LIST(IPLUS1) = LIST(I)
      LIST(I) = LTEMP
      IF(I.GT.1) J = I - 1
      GO TO 290
  320 CALL SCHDL(NMACH,TOPL,NJOB)
      J = 1
  510 TMEAN(1) = MEAN(LIST(1))
      DO 520 I=J,NJOBM1
      IPLUS1 = I + 1
      TMEAN(IPLUS1) = TMEAN(I) + MEAN(LIST(IPLUS1))
      IF((TMEAN(IPLUS1).GT.ALT1).AND.(TMEAN(I).LT.UST1)) GO TO 520
      TOPD = 0.0
      DO 293 MT=I,NJOB
  293 TOPD = TOPD + INCOM(LIST(MT))
      TOPD1 = TOPD - INCOM(LIST(I))
      IF((MEAN(LIST(IPLUS1)).GE.MEAN(LIST(I))).AND.(TOPD1.LE.TOPD))
     *GO TO 520
```

```
      CALL SCHDL(NMACH,TOPL,NJOB)
      TOTAL = TOPL
      LTEMP = LIST(IPLUS1)
      LIST(IPLUS1) = LIST(I)
      LIST(I) = LTEMP
      CALL SCHDL(NMACH,TOPL,NJOB)
      IF(TOPL.LT.TOTAL) GO TO 530
      LTEMP = LIST(IPLUS1)
      LIST(IPLUS1) = LIST(I)
      LIST(I) = LTEMP
  520 CONTINUE
      GO TO 550
  530 IF(I.GT.1) J = I - 1
      GO TO 510
  550 CALL SCHDL(NMACH,TOPL,NJOB)
      TOPL = TOPL + NMACH * WACY
      IF(TOPL.GE.DEGER) GO TO 1
      DEGER = TOPL
      IKS = NMACH
    1 CONTINUE
      COST(NODE) = CASSIG(NODE) + (DEGER / RHOPL1)
      CNODE = CASSIG(NODE) + DEGER
      CALL SCHDL(IKS,TOPL,NJOB)
  800 IF(COST(NODE).GT.UB) RETURN
      IF(IKS.EQ.0) RETURN
      CALL EVAL(IKS,CNODE)
      RETURN
  900 WRITE(6,910)
  910 FORMAT(5X,'CPU IS EXCEEDED')
      IABORT = 1
      RETURN
      END
C
C**** SUBROUTINE "PROB" COMPUTES INCOMPLETION PROBABILITIES OF TASKS
C
      SUBROUTINE PROB(T1,T2,PR1,PR2,CYC)
      PARAMETER (LN = 10000,LT = 100,LTD = 100)
      DIMENSION VAR(LT),COST(LN),CASSIG(LN),TMEAN(LT)
      REAL MEAN(LT),IC(LT),NORMAL(310),PRINC(LN,LT),INCOM(LT)
      INTEGER*2 NPRE(LT),MARK(LT),ELEMAN(LN,LT),NTNODE(LN),PARENT(LN),
     *IMPRE(LT,10),NFOL(LT),NTPRE(LT),PREC(LT,LT),
     *IMFOL(LT,10),STATUS(LT),LIST(LT),MJOB(LT),MACH(LT,LT),GSTR(LN),
     *NTFOL(LT),FOLLOW(LT,LT),IDMT(LTD),ISDF(LTD,LTD)
      COMMON INCOM,PRINC,VAR,COST,CASSIG,MEAN,IC,NORMAL,TMEAN,WACY,UB,
     *VMULT,CMULT,RHOPL1,CYCLE,ALT,UST,NTASK,LEVEL,KLSTA,NGBR,ICEL,
     *IKARA,IFLAG,IABORT,NODE,NLEVEL,NLM2,IA,MARK,NPRE,NTPRE,PREC,NTFOL,
     *FOLLOW,ISDF,IDMT,ELEMAN,GSTR,NTNODE,PARENT,IMPRE,NFOL,IMFOL,
     *STATUS,LIST,MJOB,MACH
C
      FNORM = (CYC - T1) /SQRT (T1 * VMULT)
      IF(FNORM) 110,120,120
  110 IF(FNORM.LT.-3.0) GO TO 130
      J = (0.01 - FNORM) * 100.
      PR1 = NORMAL(J)
      GO TO 200
  130 PR1 = 1.
      GO TO 200
  120 IF(FNORM.GT.3.0) GO TO 140
      J = (FNORM + 0.01) * 100.
      PR1 = 1. - NORMAL(J)
      GO TO 200
  140 PR1 = 0.0
  200 IF(T2.LT.0.0) RETURN
      FNORM = (CYC - T2) /SQRT (T2 * VMULT)
      IF(FNORM) 210,220,220
  210 IF(FNORM.LT.-3.0) GO TO 230
      J = (0.01 - FNORM) * 100.
      PR2 = NORMAL(J)
      RETURN
  230 PR2 = 1.
      RETURN
  220 IF(FNORM.GT.3.0) GO TO 240
```

```
      J = (FNORM + 0.01) * 100.
      PR2 = 1. - NORMAL(J)
      RETURN
  240 PR2 = 0.0
      RETURN
      END
C
C**** SUBROUTINE"SCHDL" SCHEDULES TASKS TO MACHINES IN
C**** RELAXED PROBLEMS
C
      SUBROUTINE SCHDL(NMACH,TOPL,NJOB)
      PARAMETER (LN=10000,LT=100,LTD=100)
      DIMENSION VAR(LT),COST(LN),CASSIG(LN),TMEAN(LT),TMACH(LT)
      REAL MEAN(LT),IC(LT),NORMAL(310),NEW,PRINC(LN,LT),INCOM(LT)
      INTEGER*2 NPRE(LT),MARK(LT),ELEMAN(LN,LT),NTNODE(LN),PARENT(LN),
     *IMPRE(LT,10),NFOL(LT),NTPRE(LT),PREC(LT,LT),
     *IMFOL(LT,10),STATUS(LT),LIST(LT),MJOB(LT),MACH(LT,LT),GSTR(LN),
     *NTFOL(LT),FOLLOW(LT,LT),IDMT(LTD),ISDF(LTD,LTD)
      COMMON INCOM,PRINC,VAR,COST,CASSIG,MEAN,IC,NORMAL,TMEAN,WACY,UB,
     *VMULT,CMULT,RHOPL1,CYCLE,ALT,UST,NTASK,LEVEL,KLSTA,NGBR,ICEL,
     *IKARA,IFLAG,IABORT,NODE,NLEVEL,NLM2,IA,MARK,NPRE,NTPRE,PREC,NTFOL,
     *FOLLOW,ISDF,IDMT,ELEMAN,GSTR,NTNODE,PARENT,IMPRE,NFOL,IMFOL,
     *STATUS,LIST,MJOB,MACH
C
      DO 595 I=1,NMACH
      TMACH(I) =0.0
  595 MJOB(I) = 0
      IF(NMACH.GT.1) GO TO 399
      DO 401 I=1,NJOB
  401 MACH(1,I) = LIST(I)
      MJOB(1) = NJOB
      GO TO 320
  399 DO 400 I=1,NJOB
      M = 1
      TDUM = TMACH(1)
      DO 500 J=2,NMACH
      IF(TMACH(J).GE.TDUM)  GO TO 500
      M = J
      TDUM = TMACH(J)
  500 CONTINUE
      MJOB(M) = MJOB(M) + 1
      MACH(M,MJOB(M)) = LIST(I)
      TMACH(M) = TMACH(M) + MEAN(LIST(I))
  400 CONTINUE
      DO 402 KK=1,NMACH
  402 CONTINUE
      DO 200 L=1,NMACH
      K = MJOB(L)
      IF(K.LE.1) GO TO 200
      DO 221 N=1,K
  221 TMEAN(N) = 0.0
      K = K - 1
      J = 1
  290 TMEAN(1) = MEAN(MACH(L,1))
      DO 300 I=J,K
      IPLUS1 = I + 1
      TMEAN(IPLUS1) = TMEAN(I) + MEAN(MACH(L,IPLUS1))
      IF(TMEAN(IPLUS1).LE.ALT)  GO TO 300
      IF(TMEAN(I).GE.UST)      GO TO 300
      MLI1 = MACH(L,IPLUS1)
      MLI = MACH(L,I)
      TOPD = 0.0
      DO 292 MT=1,MJOB(L)
  292 TOPD = TOPD + INCOM(MACH(L,MT))
      TOPD1 = TOPD - INCOM(MLI)
      IF((MEAN(MLI1).GE.MEAN(MLI)).AND.(TOPD1.LE.TOPD))  GO TO 300
      CALL PROB(TMEAN(I),TMEAN(IPLUS1),PR1,PR2,CYCLE)
      CUR = TOPD * PR1 + TOPD1 * PR2
      TMEANA = TMEAN(I)
      TMEAN(I) = TMEAN(I) - MEAN(MLI) + MEAN(MLI1)
      CALL PROB(TMEAN(I),TMEAN(IPLUS1),PR1,PR2,CYCLE)
      TOPD1 = TOPD - INCOM(MACH(L,IPLUS1))
      NEW = TOPD * PR1 + TOPD1 * PR2
      IF(CUR.GT.NEW) GO TO 310
      TMEAN(I) = TMEANA
  300 CONTINUE
      GO TO 200
  310 LTEMP = MACH(L,IPLUS1)
      MACH(L,IPLUS1) = MACH(L,I)
      MACH(L,I) = LTEMP
      IF(I.GT.1) J = I - 1
      GO TO 290
  200 CONTINUE
  320 TOPL = 0.0
      DO 600 J=1,NMACH
      T = 0.0
      K = MJOB(J)
      TOPD = 0.0
      DO 293 I=1,K
  293 TOPD = TOPD + INCOM(MACH(J,I))
      PROLD = 0.0
      DO 700 I=1,K
      T = T + MEAN(MACH(J,I))
      CALL PROB(T,-1.,PR1,PR2,CYCLE)
      TOPL = TOPL + (TOPD * (PR1 - PROLD))
      TOPD = TOPD - INCOM(MACH(J,I))
      PROLD = PR1
  700 CONTINUE
  600 CONTINUE
      RETURN
      END
C
C**** SUBROUTINE "EVAL" CHECKS IF THE NODE IS FEASIBLE
C
      SUBROUTINE EVAL(IKS,CNODE)
      PARAMETER (LN=10000,LT=100,LTD=100)
      DIMENSION VAR(LT),COST(LN),CASSIG(LN),TMEAN(LT),GAM(LT,LT)
      REAL MEAN(LT),IC(LT),NORMAL(310),PRINC(LN,LT),INCOM(LT)
      INTEGER*2 NPRE(LT),MARK(LT),ELEMAN(LN,LT),NTNODE(LN),PARENT(LN),
     *IMPRE(LT,10),NFOL(LT),NTPRE(LT),PREC(LT,LT),
     *IMFOL(LT,10),STATUS(LT),LIST(LT),MJOB(LT),MACH(LT,LT),
     *GSTR(LN),KES(LT),NTFOL(LT),FOLLOW(LT,LT),IYEN(LT),LABEL(LT),
     *IDMT(LTD),ISDF(LTD,LTD)
      COMMON INCOM,PRINC,VAR,COST,CASSIG,MEAN,IC,NORMAL,TMEAN,WACY,UB,
     *VMULT,CMULT,RHOPL1,CYCLE,ALT,UST,NTASK,LEVEL,KLSTA,NGBR,ICEL,
     *IKARA,IFLAG,IABORT,NODE,NLEVEL,NLM2,IA,MARK,NPRE,NTPRE,PREC,NTFOL,
     *FOLLOW,ISDF,IDMT,ELEMAN,GSTR,NTNODE,PARENT,IMPRE,NFOL,IMFOL,
     *STATUS,LIST,MJOB,MACH
C
      DO 10 I=1,NTASK
      KES(I) = 0
      IF(STATUS(I).NE.1) GO TO 12
      KES(I) = 1
      GO TO 10
   12 M = NTNODE(NODE)
      DO 15 J=1,M
      IF(ELEMAN(NODE,J).NE.I) GO TO 15
      KES(I) = 1
      GO TO 10
   15 CONTINUE
   10 CONTINUE
      DO 20 I=1,IKS
      NJ = MJOB(I)
      DO 30 J=1,NJ
      NK = MACH(I,J)
      KP = NPRE(NK)
      DO 40 L=1,KP
      IF(KES(IMPRE(NK,L)).EQ.0) RETURN
   40 CONTINUE
      KES(NK) = 1
   30 CONTINUE
   20 CONTINUE
      GSTR(NODE) = 1
      MLEVEL = LEVEL
      MNODE = NODE
      TFIAT = 0.0
      IHIY = 30
      DO 100 I=1,IKS
      PARENT(NODE+1) = NODE
      NODE = NODE + 1
      NTNODE(NODE) = MJOB(I)
      DO 101 IH=1,MJOB(I)
  101 ELEMAN(NODE,IH) = MACH(I,IH)
      LEVEL = LEVEL + 1
      CALL COSCAL(FIAT)
      TFIAT = TFIAT + FIAT
  100 CONTINUE
      NODE = MNODE
      TFIAT = TFIAT + CASSIG(NODE) + (WACY * FLOAT(IKS))
      IF(TFIAT.GE.UB) GO TO 899
      CALL TIMECK(NTIME)
      TIME = NTIME / 100.
      WRITE(6,900) NODE,TFIAT,TIME
      UB = TFIAT
      L = NODE
      KK = 1
      DO 60 I=1,NTASK
      IF(PARENT(L).EQ.0) GO TO 65
      KK = KK + 1
      L = PARENT(L)
   60 CONTINUE
   65 L = NODE
      DO 66 I=1,KK
      KNE = KK - I + 1
      WRITE(6,910) KNE,(ELEMAN(L,KS),KS=1,NTNODE(L))
   66 L = PARENT(L)
      DO 70 I=1,IKS
      KNE = KK + I
   70 WRITE(6,910) KNE,(MACH(I,J),J=1,MJOB(I))
      DO 50 I=IA,NODE
      IF(GSTR(I).EQ.1) GO TO 50
      IF(COST(I).GT.UB) GSTR(I) = 1
   50 CONTINUE
  899 DO 102 I=1,IKS
      PARENT(NODE+1) = 0
      NODE = NODE + 1
      NTNODE(NODE) = 0
      DO 103 IH=1,MJOB(I)
      ELEMAN(NODE,IH) = 0
  103 PRINC(NODE,IH) = 0.0
  102 CONTINUE
      NODE = MNODE
      LEVEL= MLEVEL
      RETURN
  900 FORMAT(//,8X,'NODE',I6,' IS A FEASIBLE SOLUTION',/,8X,
     *'NEW UPPER BOUND =  ',F8.4,/,8X,
     *'CPU TIME SPENT TO OBTAIN THE NODE =  ',F8.3,
     *//,8X,'DESIGN OF THE SOLUTION',/)
  910 FORMAT(5X,'STATION :',I3,3X,'SEQUENCE :',20(I3,','))
      END
C
C
      FUNCTION MATRIX(I,J)
      MATRIX = 0
      IF(J.EQ.1) GO TO 10
      K = ( I / ( INT ( 2 ** ( J - 1 )))) + 1
      IF(MOD(K,2).EQ.1) RETURN
      MATRIX = 1
      RETURN
   10 IF(MOD(I,2).NE.1) RETURN
      MATRIX = 1
      RETURN
      END
C
C
C
```

```
      FUNCTION MATCOM(I,J)
      MATCOM = 0
      IF(I.EQ.1) RETURN
      IF(J.EQ.1) GO TO 10
      K = ( (I-1) / ( INT ( 2 ** ( J - 1 )))) + 1
      IF(MOD(K,2).EQ.1) RETURN
      MATCOM = 1
      RETURN
   10 IF(MOD(I+1,2).NE.1) RETURN
      MATCOM = 1
      RETURN
      END
C
C
      SUBROUTINE PRCMB(SANCOM)
      PARAMETER (LN=10000,LT=100,LTD=100,LDAL=2000)
      DIMENSION VAR(LT),COST(LN),CASSIG(LN),TMEAN(LT)
      REAL MEAN(LT),IC(LT),NORMAL(310),PRINC(LN,LT),PONCE(LDAL),
     *DMEAN(LDAL),DVAR(LDAL),DSANS(LDAL),INCOM(LT)
      INTEGER*2 NPRE(LT),MARK(LT),ELEMAN(LN,LT),NTNODE(LN),PARENT(LN),
     *IMPRE(LT,10),NFOL(LT),NTPRE(LT),PREC(LT,LT),
     *IMFOL(LT,10),STATUS(LT),LIST(LT),MJOB(LT),MACH(LT,LT),GSTR(LN),
     *NTFOL(LT),FOLLOW(LT,LT),NST(LT),LABEL(LDAL),FINISH(LT),
     *NSTART(LT,LT),IS(LT,LT),NIS(LT),INFO(LDAL,LT),DAL,BNO,FINO,
     *IDMT(LTD),ISDF(LTD,LTD)
      COMMON INCOM,PRINC,VAR,COST,CASSIG,MEAN,IC,NORMAL,TMEAN,WACY,UB,
     *VMULT,CMULT,RHOPL1,CYCLE,ALT,UST,NTASK,LEVEL,KLSTA,NGBR,ICEL,
     *IKARA,IFLAG,IABORT,NODE,NLEVEL,NLM2,IA,MARK,NPRE,NTPRE,PREC,NTFOL,
     *FOLLOW,ISDF,IDMT,ELEMAN,GSTR,NTNODE,PARENT,IMPRE,NFOL,IMFOL,
     *STATUS,LIST,MJOB,MACH
      COMMON /EKLEME/BNO,FINO,NST,NSTART,FINISH
C
      KC = PARENT(NODE)
      NSTM1 = LEVEL - 1
      DO 100 I=1,NSTM1
      DO 120 J=1,NTNODE(KC)
  120 IS(NSTM1-I+1,J) = ELEMAN(KC,J)
      NIS(NSTM1-I+1) = NTNODE(KC)
  100 KC = PARENT(KC)
      DO 110 I=1,NTASK
  110 INFO(1,I) = 0
      DO 111 I=1,LDAL
  111 LABEL(I) = 0
      DAL = 1
      DSANS(1) = 1.0
      DMEAN(1) = 0.0
      DVAR(1) = 0.0
      IESKI = 1
      IYENI = 1
      PONCE(1) = 1.0
      DO 130 I=1,NSTM1
      DO 140 J=1,NIS(I)
      ISIJ = IS(I,J)
      DO 160 IZMIR=IESKI,IYENI
      IF(LABEL(IZMIR).EQ.1) GO TO 160
      IF(NTPRE(ISIJ).EQ.0) GO TO 162
      DO 165 IB=1,NTPRE(ISIJ)
      IF(INFO(IZMIR,PREC(ISIJ,IB)).EQ.0) GO TO 200
  165 CONTINUE
  162 DAL = DAL + 1
      DO 170 IB=1,NTASK
  170 INFO(DAL,IB) = INFO(IZMIR,IB)
      IF(J.EQ.1) THEN
      PONCE(DAL) = DSANS(IZMIR)
      DMEAN(DAL) = MEAN(ISIJ)
      DVAR(DAL) = VAR(ISIJ)
      ELSE
      PONCE(DAL) = PONCE(IZMIR)
      DMEAN(DAL) = DMEAN(IZMIR) + MEAN(ISIJ)
      DVAR(DAL) = DVAR(IZMIR) + VAR(ISIJ)
      ENDIF
      INFO(DAL,ISIJ) = 1
      IF(DVAR(DAL).LE.0.0)  DVAR(DAL) = 0.001
      FNORM = (CYCLE - DMEAN(DAL)) / SQRT(DVAR(DAL))
      CALL PRCA2(DPROB, FNORM)
      DSANS(DAL) = (1. - DPROB) * PONCE(DAL)
      IF(DSANS(DAL).LE.0.0)  LABEL(DAL) = 1
      DSANS(DAL + 1) = DSANS(IZMIR) - DSANS(DAL)
      DAL = DAL + 1
      IF(DSANS(DAL).LE.0.0)  LABEL(DAL) = 1
      GO TO 205
  200 DAL = DAL + 1
      DSANS(DAL) = DSANS(IZMIR)
  205 DO 175 IB = 1,NTASK
      INFO(DAL,IB) = INFO(IZMIR,IB)
  175 CONTINUE
      IF(J.EQ.1) THEN
      PONCE(DAL) = DSANS(IZMIR)
      ELSE
      PONCE(DAL) = PONCE(IZMIR)
      ENDIF
      DO 180 IB = 1,FINO
      IF(FINISH(IB).EQ.ISIJ)  LABEL(DAL) = 1
  180 CONTINUE
  160 CONTINUE
      IESKI = IYENI + 1
      IYENI = DAL
  140 CONTINUE
  130 CONTINUE
      SANCOM = 0.0
      DO 206 I = IESKI,IYENI
      IF(LABEL(I).EQ.1)  GO TO 206
      IF(BNO.EQ.0)  GO TO 260
      DO 210 J = 1,BNO
      DO 250 IZMIR = 1,NST(J)
      IF(INFO(I,NSTART(J,IZMIR)).EQ.0)  GO TO 210
  250 CONTINUE
      GO TO 206
  210 CONTINUE
  260 SANCOM = SANCOM + DSANS(I)
  206 CONTINUE
      RETURN
      END
C
C
      SUBROUTINE PRCA2(SUAT,FNORM)
      PARAMETER (LN = 10000,LT = 100,LTD = 100)
      DIMENSION VAR(LT),COST(LN),CASSIG(LN),TMEAN(LT)
      REAL MEAN(LT),IC(LT),NORMAL(310),PRINC(LN,LT),INCOM(LT)
      INTEGER*2 NPRE(LT),MARK(LT),ELEMAN(LN,LT),NTNODE(LN),PARENT(LN),
     *IMPRE(LT,10),NFOL(LT),NTPRE(LT),PREC(LT,LT),
     *IMFOL(LT,10),STATUS(LT),LIST(LT),MJOB(LT),MACH(LT,LT),GSTR(LN),
     *NTFOL(LT),FOLLOW(LT,LT),IDMT(LTD),ISDF(LTD,LTD)
      COMMON INCOM,PRINC,VAR,COST,CASSIG,MEAN,IC,NORMAL,TMEAN,WACY,UB,
     *VMULT,CMULT,RHOPL1,CYCLE,ALT,UST,NTASK,LEVEL,KLSTA,NGBR,ICEL,
     *IKARA,IFLAG,IABORT,NODE,NLEVEL,NLM2,IA,MARK,NPRE,NTPRE,PREC,NTFOL,
     *FOLLOW,ISDF,IDMT,ELEMAN,GSTR,NTNODE,PARENT,IMPRE,NFOL,IMFOL,
     *STATUS,LIST,MJOB,MACH
C
      IF(FNORM) 110,120,120
  110 IF(FNORM.LT.-3.0)  GO TO 130
      J = (0.01 - FNORM) * 100.
      SUAT = NORMAL(J)
      RETURN
  130 SUAT = 1.
      RETURN
  120 IF(FNORM.GT.3.0) GO TO 140
      J = (FNORM + 0.01) * 100.
      SUAT = 1. - NORMAL(J)
      RETURN
  140 SUAT = 0.0
      RETURN
      END
C
C
      SUBROUTINE COSCAL(FIAT)
      PARAMETER (LN = 10000,LT = 100,LTD = 100)
      DIMENSION VAR(LT),COST(LN),CASSIG(LN),TMEAN(LT)
      REAL MEAN(LT),IC(LT),NORMAL(310),PRINC(LN,LT),INCOM(LT)
      INTEGER*2 NPRE(LT),MARK(LT),ELEMAN(LN,LT),NTNODE(LN),PARENT(LN),
     *IMPRE(LT,10),NFOL(LT),NTPRE(LT),PREC(LT,LT),COM(LT),LABEL(LT),
     *IMFOL(LT,10),STATUS(LT),LIST(LT),MJOB(LT),MACH(LT,LT),GSTR(LN),
     *NTFOL(LT),FOLLOW(LT,LT),NST(LT),NSTART(LT,LT),FINISH(LT),FINO,BNO,
     *IDMT(LTD),ISDF(LTD,LTD)
      COMMON INCOM,PRINC,VAR,COST,CASSIG,MEAN,IC,NORMAL,TMEAN,WACY,UB,
     *VMULT,CMULT,RHOPL1,CYCLE,ALT,UST,NTASK,LEVEL,KLSTA,NGBR,ICEL,
     *IKARA,IFLAG,IABORT,NODE,NLEVEL,NLM2,IA,MARK,NPRE,NTPRE,PREC,NTFOL,
     *FOLLOW,ISDF,IDMT,ELEMAN,GSTR,NTNODE,PARENT,IMPRE,NFOL,IMFOL,
     *STATUS,LIST,MJOB,MACH
      COMMON /EKLEME/BNO,FINO,NST,NSTART,FINISH
C
      NJOB = NTNODE(NODE)
      FIAT = 0.0
      DO 90 L = 1,NJOB
      IDKL = ELEMAN(NODE,L)
      CPAR = 0.0
      IF(L.EQ.1) GO TO 100
      LMNS1 = L - 1
      INDF = 2 ** LMNS1
      I = 1
  105 CONTINUE
      DO 120 J = 1,LMNS1
      IF(MATCOM(I,J).EQ.1)  THEN
      COM(J) = 1
      ELSE
      COM(J) = 0
      ENDIF
  120 CONTINUE
      COM(L) = 0
      FINO = 0
      DO 130 J = 1,L
      IF(COM(J).EQ.1)  GO TO 130
      IDKJ = ELEMAN(NODE,J)
      DO 150 IZMIR = 1,NTASK
      DO 160 IG = 1,J
      IF(IZMIR.EQ.ELEMAN(NODE,IG))  GO TO 150
  160 CONTINUE
      IF(FINO.EQ.0)  GO TO 156
      DO 155 IH = 1,FINO
      IF(IZMIR.EQ.FINISH(IH))  GO TO 150
  155 CONTINUE
  156 DO 170 IG = 1,NTPRE(IDKJ)
      IF(PREC(IDKJ,IG).NE.IZMIR)  GO TO 170
      FINO = FINO + 1
      FINISH(FINO) = IZMIR
      GO TO 150
  170 CONTINUE
  150 CONTINUE
  130 CONTINUE
  987 BNO = 0
      DO 200 J = 1,LMNS1
      IF(COM(J).EQ.0)  GO TO 200
      IDKJ = ELEMAN(NODE,J)
      IF(NTPRE(IDKJ).EQ.0)  GO TO 110
      BNO = BNO + 1
      NST(BNO) = 0
      DO 210 IZMIR = 1,NTASK
      DO 220 IB = 1,J
      IF(IZMIR.EQ.ELEMAN(NODE,IB))  GO TO 210
  220 CONTINUE
      DO 230 IB = 1,FINO
      IF(IZMIR.EQ.FINISH(IB))  GO TO 210
  230 CONTINUE
      IF(NST(BNO).EQ.0)  GO TO 245
```

```
    DO 246 IH=1,NST(BNO)
    IF(IZMIR.EQ.NSTART(BNO,IH))  GO TO 210
246 CONTINUE
245 DO 240 IB=1,NTPRE(IDKJ)
    IF(PREC(IDKJ,IB).NE.IZMIR)  GO TO 240
    NST(BNO) = NST(BNO) + 1
    NSTART(BNO,NST(BNO)) = IZMIR
240 CONTINUE
210 CONTINUE
988 IF(NST(BNO).EQ.0)  GO TO 110
200 CONTINUE
    GO TO 400
100 FINO = 0
    BNO = 0
    COM(L) = 0
    IF(NTPRE(IDKL).EQ.0)  GO TO 400
    DO 300 IZMIR=1,NTASK
    DO 310 IB=1,NTPRE(IDKL)
    IF(PREC(IDKL,IB).NE.IZMIR)  GO TO 310
    FINO = FINO + 1
    FINISH(FINO) = IZMIR
    GO TO 300
310 CONTINUE
300 CONTINUE
    IF(FINO.EQ.0) GO TO 400
400 IF(LEVEL.GT.1)  GO TO 410
    SANCOM = 1.0
    GO TO 420
410 CALL PRCMB(SANCOM)
420 TOPLM = 0.0
    TOPLV = 0.0
    DO 415 J=1,L
    IF(COM(J).EQ.1)  GO TO 415
    TOPLMO = TOPLM
    TOPLVO = TOPLV
    TOPLM = TOPLM + MEAN(ELEMAN(NODE,J))
    TOPLV = TOPLV + VAR(ELEMAN(NODE,J))
415 CONTINUE
    IF(TOPLMO.GT.0.0)  GO TO 604
    SANONC = 0.0
    GO TO 605
604 IF(TOPLVO.LE.0.0)  TOPLVO =  0.001
    FNORMO = (CYCLE - TOPLMO) / SQRT(TOPLVO)
    CALL PRCA2(SANONC,FNORMO)
605 IF(TOPLV.LE.0.0)  TOPLV =  0.001
    FNORM = (CYCLE - TOPLM) / SQRT(TOPLV)
    CALL PRCA2(SANTAK,FNORM)
    SANGEC = (SANTAK - SANONC) * SANCOM
    PRINC(NODE,L) = PRINC(NODE,L) + SANGEC
    KC = NODE
    DO 550 IZMIR=1,LEVEL
    IF(IZMIR.EQ.1)  GO TO 555
    DO 560 IB=1,NTNODE(KC)
    CTOP = 0.0
    IDKCIB = ELEMAN(KC,IB)
    IF(FINO.EQ.0)  GO TO 566
    DO 565 IG=1,FINO
    IF(IDKCIB.EQ.FINISH(IG))  GO TO 560
565 CONTINUE
566 DO 570 IG=1,NTASK
570 LABEL(IG) = 0
    DO 585 IXY=IB,NTNODE(KC)
    IDX = ELEMAN(KC,IXY)
    DO 580 IG=1,NTFOL(IDX)
580 LABEL(FOLLOW(IDX,IG)) = 1
585 CONTINUE
    DO 591 IXY=L,NJOB
    IDX = ELEMAN(NODE,IXY)
    DO 586 IG=1,NTFOL(IDX)
    IF(LABEL(FOLLOW(IDX,IG)).EQ.1) LABEL(FOLLOW(IDX,IG))=2
586 CONTINUE
591 CONTINUE
```

```
      DO 592 IXY=1,NTASK
      IF(LABEL(IXY).EQ.2) CTOP = CTOP + INCOM(IXY)
  592 CONTINUE
      CPAR = CPAR + (CTOP * PRINC(KC,IB) * SANGEC)
  560 CONTINUE
  555 KC = PARENT(KC)
  550 CONTINUE
  416 IF(L.EQ.1) GO TO 108
  110 I = I + 1
      IF(I.LE.INDF) GO TO 105
  108 CONTINUE
      IF(IFLAG.EQ.0) THEN
      DO 600 I=1,NTASK
  600 LABEL(I) = 0
      DO 610 I=L,NJOB
      IDKI = ELEMAN(NODE,I)
      LABEL(IDKI) = 1
      DO 620 IZMIR=1,NTFOL(IDKI)
      LABEL(FOLLOW(IDKI,IZMIR)) = 1
  620 CONTINUE
  610 CONTINUE
      CIC = 0.0
      DO 630 I=1,NTASK
      IF(LABEL(I).EQ.1) CIC = CIC + INCOM(I)
  630 CONTINUE
      FIAT = FIAT + (PRINC(NODE,L) * CIC) - CPAR
      ELSE
      DO 1600 I=1,ICEL
 1600 LABEL(I) = 0
      DO 1610 I=L,NJOB
      IDKI = ELEMAN(NODE,I)
      LABEL(IDKI) = 1
      DO 1620 IZMIR=1,IDMT(IDKI)
      LABEL(ISDF(IDKI,IZMIR)) = 1
 1620 CONTINUE
 1610 CONTINUE
      CIC = 0.0
      DO 1630 I=1,ICEL
      IF(LABEL(I).EQ.1) CIC = CIC + INCOM(I)
 1630 CONTINUE
      FIAT = FIAT + (PRINC(NODE,L) * CIC) - CPAR
      ENDIF
   90 CONTINUE
      RETURN
      END
```