A Computerized Methodology for Balancing and Sequencing

Mixed Model Stochastic Assembly Lines

by

John P. Pantouvanos

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

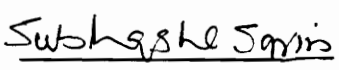
MASTER OF SCIENCE

in

Industrial and Systems Engineering

APPROVED:

Osama K. Eyada, Chairman



Subhash C. Sarin

Robert T. Sumichrast

January 1992

Blacksburg, Virginia

A COMPUTERIZED METHODOLOGY FOR BALANCING AND SEQUENCING MIXED MODEL STOCHASTIC ASSEMBLY LINES

by

John P. Pantouvanos

Committee Chairman: Osama K. Eyada

Industrial and Systems Engineering

(ABSTRACT)

A methodology for designing mixed model stochastic assembly line systems and a computer package to implement it for realistic problem sizes were developed. The methodology consists of three major steps: (1) generation of feasible sequences of feeding models into the line, (2) generation of feasible allocations (balances) of work elements to work stations, and (3) generation of combinations of sequences and allocations with the best ordering of elements within stations, calculation of total expected cost for each combination, and selection of the one with the least cost.

For generating feasible balances, an exhaustive search procedure with a number of heuristic rules was used to ensure searching the whole feasible region in limited time. A cost model based on labor and incompletion costs is used to calculate the cost of each combination, and a recursive procedure to calculate incompletion probabilities for each element and incorporate them into the cost model was implemented. An example problem, its results, and the computer package listings are included.

ACKNOWLEGMENTS

I am indebted to the chairman of my examining committee Dr. Osama Eyada and the members Dr. Subhash Sarin and Dr. Robert Sumichrast for their time, guidance and wisdom. It has been a pleasant and fruitful experience working with them.

I wish to thank all the teachers I ever had, who, throughout the years, shaped my beliefs, made me who I am, and all helped in making me wiser.

Regarding this thesis, I would like to thank my very good friend Stelios Corres for industriously proofreading parts of the text at various stages and for his valuable suggestions as to its style and format. Special thanks to the fine programmer and very good friend of mine Andreas Stokas who helped me master the C programming language and who was the person who played the biggest role in my decision for graduate studies. This thesis would have not be accomplished without a little help from my friends, this is why I wish to thank them all, particularly for their encouragement.

Finally, I would like to take the opportunity to express my appreciation and gratitude to my parents for all they have ever done for me, and for passing on to me their ethics and values.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

K = number of work elements

J = number of models

n = number of stations in the current solution

M = number of products in a reduced sequence

N_j = scheduled quantity of model j, $N_j \in N^+, 1 \leq j \leq J$

bcd = the biggest common divisor of the scheduled quantities per shift for all models

t_k = actual processing time for element k, $1 \leq k \leq K$

$\hat{t}_k$ = expected processing time for element k, $1 \leq k \leq K$

twc = average total work content per part for all parts scheduled during a shift

twc_i = average total work content per part for the first i parts scheduled during a shift

c = fixed launching rate

a = operator allowed time

d = time interval (delay) between a part entering a station and the operator starting processing this part

d_1 = delay due to work of the previous operator on the current part

d_2 = delay due to work of the current operator on the previous part

p_{ij} = expected assigned time to station i for each product of model j, $1 \leq i \leq n, 1 \leq j \leq J$

Prob *(e)* = Probability of event e happening

$F_{t_1,t_2,\ldots,t_i}(x) = Prob\,(t_1+t_2+\ldots+t_i \leq x)$ = Cumulative distribution function of times $t_1,t_2,\ldots,t_i$

i_k = expected occurences of incompletion for element k, $1 \leq k \leq K$ during a shift

L = uniform labor rate per operator per cycle time unit

I_k = incompletion cost of element k for each occurrence of incompletion, $1 \leq k \leq K$

$IC^{k,m}_i$ = Expected incompletion cost of element i at station k for part m

EC = Total expected cost for a combination of balance and sequence

Q^m_k = Set of elements allowed to occur each time for part m at station k

R_{k-1} = Set of elements assigned to stations 1 through k-1

r = history of a part = set of finished elements at each point

P_i = Set of elements that are precedents of element i

D_i = Set of elements assigned to the same station as element i after i, and all of i's followers

CHAPTER 1: INTRODUCTION

1.1. Types and Characteristics of Assembly Lines

The majority of industrial products requires assembly during manufacturing. Product assembly costs can range between 25 and 50 percent of total costs (Kalpakjian 1989). Efficient design of assembly operations or lines is therefore an important factor for designing manufacturing systems. In a mass production environment, assembly operations are performed on assembly lines where the total assembly work is divided among stations, and the product is manually or mechanically fed from one station to the next.

Traditionally, assembly lines were devoted to the mass production of a single product. This is known as **single model assembly.** With the current customers' trend toward customized and specialized products, the necessity of assembling different models of the same product on one line prevailed. One way to handle multiple models on the same line is to schedule their production in batches. This is the approach of **batch assembly,** and it is usually utilized when some setup work is needed to switch from one model to the next. Another approach is the **mixed model assembly,** where different models are launched on the assembly line without the need for any setup changes. The main advantage of this approach over batch assembly is the increased flexibility in responding to demand fluctuations.

Assembly lines can take many forms depending upon the: (1) conditions prevailing during assembly, such as paced or unpaced transfer mechanism, closed or open assembly station and degree of automation, (2) associated costs including labor, equipment and incompletion costs, (3) available space, manpower and equipment, and (4) the type of elementary task times (deterministic or probabilistic). If

the work performed at an assembly station is totally automated, the time needed to finish an assembly task is **deterministic** or, seldom, probabilistic with very low variability. When stations are manned by human operators, or operators play a crucial role in the operation of equipment, the task time is **probabilistic.**

Examination of the literature indicates that probabilistic task times are referred to as **stochastic,** although they do not depend on time. They just follow a probability distribution function, which can be approximated by continuous distributions (e.g., normal or uniform). For the majority of assembly lines, stochastic times occur more often than deterministic.

1.2. Design of Assembly Lines

There are many factors that can influence the efficiency of operating an assembly line to meet product demands and projected costs. These include number of operators, type of product flow (paced or unpaced), type of station (open or closed), and whether parallel or enlarged (enriched) stations will be used.

Once these factors have been determined for a mixed model assembly line, two important steps must be performed for the design of such a system: **Assembly Line Balancing** and **Assembly Line Sequencing.** In assembly line balancing, the various assembly tasks that need to be performed are distributed to assembly stations. In assembly line sequencing, the appropriate sequence of models including their sequence to meet product demands is determined.

Assembly line balancing influences the actual productivity of the line and/or the number of operators needed. An ideal balance is the one where all operators are assigned the same amount of

work, and all produce with the same production rate. A realistic balance distributes tasks to operators as evenly as possible, and guarantees that the line will operate at minimum cost.

Assembly line sequencing determines whether additional inefficiencies such as idle time of operators or product will occur during assembly. An ideal sequence is the one that ensures that the production rate is the same as determined by the corresponding balance. A realistic sequence is one that ascertains as little idle time and work congestion as possible for a given balance.

Several constraints have to be considered when designing an assembly line. The most elementary ones are precedence constraints between elementary tasks, and constraints dealing with the desirable minimum production rate, available operators and duration of the shift.

1.3. Mixed Model Assembly Lines

1.3.1. Line Balancing

In comparison with the single model assembly line balancing problem, very little work has been done on the mixed model. For the deterministic case, Wester and Kilbridge (1964) are the first to propose a heuristic procedure for balancing each model separately. Later, Thomopoulos introduced a method for forcing elements to be performed at the same station for all models (1967), and an objective function for effective balancing of the line among and within models (1970). A more recent work is by Smith (1990), where he proposed an exhaustive approach which combined Thomopoulos' objective function with a number of heuristics.

For the stochastic case, Johnson (1983) outlined some considerations for addressing the

problem. Vrat and Virani (1976) analyzed a stochastic case from a very narrow perspective and with an extremely large number of models.

1.3.2. Line Sequencing

For the deterministic case, Wester and Kilbridge's work (1964) included some heuristics for obtaining a sequence that would minimize inefficiencies for only one of the models. Thomopoulos (1967) modified slightly the same procedure under different assumptions. Another heuristic was introduced by Dar-El and Cother (1975), and modified later (Dar-El and Cucuy 1977) to include an exact procedure that minimizes idle product time without any additional operator idle time. Smith (1990) proposed an exhaustive search procedure with the objective of minimizing the assembly line length. All the above methods consider a fixed balance and produce a sequence for that particular balance.

For the stochastic case, the only important work is that of Okamura and Yamashina (1979). The problem they addressed is the minimization of the risk of interrupting the production flow during assembly because of incomplete tasks. However, one of their assumptions calls for stopping the line until an incomplete task is processed, which is not an acceptable industrial practice. Also, the variability of the task times has not been included in their procedure.

1.4. Problem Definition

From the above, it can be concluded that the stochastic mixed model assembly problem has not been fully addressed in terms of balancing and sequencing. Also, no effort has been made to consider balancing and sequencing together. It is not necessarily true that the best balance found,

when combined with the best sequence for this balance, will produce the most inexpensive overall line design.

This thesis focuses on the mixed model stochastic line balancing and sequencing problem with the objective of minimizing the total cost over a shift. The problem is treated exhaustively, with the addition of heuristics for the case of large size, realistic problems. The solution sought is a combination of balancing and sequencing, with an overall minimum cost under some assumptions described in Chapter 3.

1.5. Research Objectives

The objective of this thesis is the development of a user-friendly, modular computer package to treat the above problem in two parts: Balancing and Sequencing. Exhaustive search procedures in combination with heuristic rules are used for both parts, with a focus on capturing all aspects of the problem, and producing a solution with an expected cost very close to the optimum within a reasonable time frame. The programs generate multiple combinations of balances and sequences with very close costs to the optimum, and propose the best overall combination of balance-sequence.

1.6. Thesis Outline

The thesis is organized as follows: Chapter 2 provides a literature review of relevant works. Chapter 3 describes the proposed methodology. The results of the work and their analysis will be presented in Chapter 4. Finally, conclusions and recommendations for future work are given in Chapter 5.

CHAPTER 2: LITERATURE REVIEW

Figure 1 shows a classification of existing works. Articles on balancing or sequencing may concern the **single** or **mixed model,** and may address **deterministic** or **stochastic times.**

2.1. Assembly Line Balancing

2.1.1. Single Model Deterministic Problem

Salveson (1955) was the first to publish on assembly line balancing. He brought to light some of the important aspects of the problem, and proposed a method for generating feasible balances. However, his method is short from being an algorithm.

Mathematical formulation of the problem with a corresponding procedure for solving it was investigated by four different approaches. **Bowman** (1960) formulated the problem as a linear programming model. **Held et al.** (1963) used a dynamic programming formulation. **Klein** (1963) utilized a network representation. **Betts** and **Mahmoud** (1989a) employed a branch and bound algorithm, and modified it to generate multiple solutions instead of a single one (1989b). However, these attempts cannot be utilized for solving problems of typical sizes, and require considerable time and calculations in order to reach an optimum.

It was this realization that the problem cannot be efficiently solved exactly that set off the development of heuristics in 1961. The most important heuristic procedures developed for the single model, deterministic problem are summarized in Table 1.

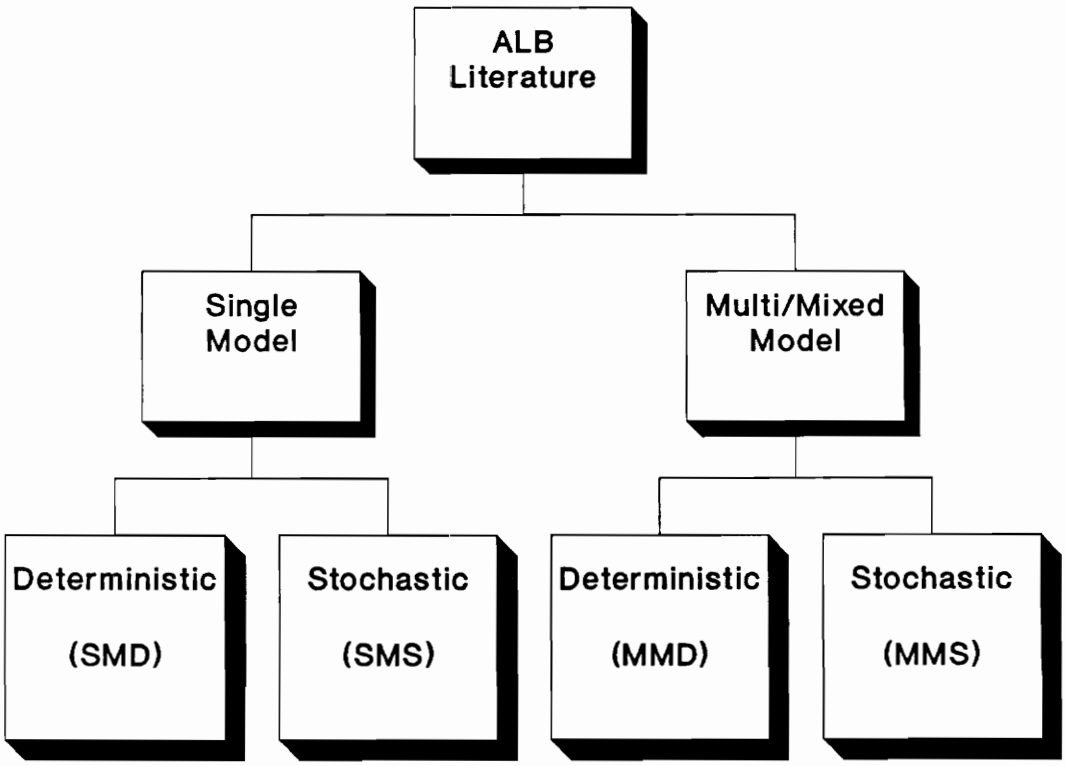


Figure 1. Classification of Assembly Line Balancing Literature

Kilbridge and **Wester** (1961) proposed a procedure for producing a good enough line balance, which works well with scarce precedence matrices and with comparably low elements to stations ratios. Their method depends highly on the person performing the balancing and cannot properly be called an algorithm, since there is no guarantee of consistent results. There is also no guarantee as to how near optimal the solutions are.

Helgeson and **Birnie** (1961) proposed the ranked positional weight technique. In this method, an attempt is made to place elements with a long time and long lasting followers as early in the line as possible. The resulting solutions have overloaded early stations and a lot of idle time for the following ones, and therefore are not guaranteed to be near the optimum. Nonetheless, the technique remains popular since it is very easy to carry out.

All the above approaches had a sole objective: to minimize the number of operators for a given production rate. Other considerations were only theoretically discussed and none of them was part of a workable algorithm. **Tonge** (1960, 1961) was the first researcher to consider the possibility of having a dual objective, few operators as well as high productivity (equal service time among stations). His work consisted of reducing the problem to balancing subsets of tasks, and then regrouping these subsets into a feasible balance while keeping them intact if possible. For the first time, using the computer as a tool was seriously considered. The problem with this approach, however, is that a significant number of feasible solutions are overlooked since it is not an exhaustive search.

Giving emphasis on the computationally expensive nature of the problem, **Dar-El** (1973) proposed an exact backtracking algorithm. It is based on element substitutions with imposed heuristic rules. The rules are geared toward reducing the feasible interval for each station's service time, and limiting the number of substitutions within and among stations and overall.

Table 1. Heuristic Procedures for the Single Model Deterministic Problem

Author(s)	Year	Article	Contribution	Characteristics	Shortcomings
1. Kilbridge, Wester	1961	A Heuristic Method of Assembly Line Balancing.	First serious effort for large problems.	Heuristic Procedure. Distribution of elements according to precedences into columns, and then stations.	Not implementable on a computer. Does not guarantee good results.
2. Helgeson, Birnie	1961	Assembly Line Balancing Using the Ranked Positional Weight Technique.	Easy to follow and implement in a computer.	Heuristic Algorithm. Assignments of elements with high positional weight into early stations.	Does not guarantee good results. Overloads early stations.
3. Tonge	1961	A Heuristic Program for Assembly Line Balancing.	Dual objective: few operators and high productivity. Serious attempt of solving large problems with a computer.	Heuristic Procedure. Identify subsets of tasks, balance these subsets, making as few substitutions as possible between subsets.	Extra imposed constraints speed up solution, but may generate inferior solutions.
4. Dar-El	1973	MALB - A Heuristic Technique for Balancing Large Single-Model Assembly Lines.	Efficient algorithm to deal with large problems.	Heuristic Algorithm. Limit to the number of interchanging elements while generating a feasible solution.	Only a small proportion of randomly selected feasible solutions are investigated.

2.1.2. Mixed Model Deterministic Problem

The major research works on the mixed model deterministic problem are summarized in Table 2.

The first work on balancing and sequencing the mixed model assembly line was by **Wester** and **Kilbridge** (1964). Their approach was to balance each model separately for a fixed number of stations, and then to group all models in the same assembly line. **Thomopoulos** (1967) modified the heuristic to force similar elements to be processed in the same station for all models, assuming that each element required the same amount of time for all models. With this procedure, he really took advantage of the mixed model situation, since expensive equipment does not have to be duplicated, and operators do not have to specialize in too many tasks. In a later article, **Thomopoulos** (1970) introduced a new objective function for line balancing, which minimizes the difference in work contents between stations as well as models to minimize product and operator idle times.

2.1.3. Single Model Stochastic Problem

The most important articles on the single model stochastic problem are summarized in Table 3.

Moodie and **Young** (1965) were the first to consider the stochastic nature of assembly task times. They proposed a two phase procedure, based on normally distributed task times. In the first phase, an initial, possibly infeasible balance is created. In the second phase, some of the elements are rearranged to produce a feasible, smoother balance. However, not all feasible solutions are investigated and the proposed solution is not guaranteed to be near the optimum.

Table2. Work on the Mixed Model Deterministic Problem

Author(s)	Year	Article	Contribution	Characteristics	Shortcomings
1. Wester, Kilbridge	1964	The Assembly Line Model-Mix Sequencing Problem.	First work on Mixed Model addressing Balancing and Sequencing.	Balancing: Independent balancing of each model. Sequencing: Procedures for variable and fixed rate launching.	Balancing: Same elements possibly assigned to more than one station for different models. Sequencing: Optimization over only one model at a time.
2. Thomopoulos	1967	Line Balancing-Sequencing for Mixed-Model Assembly.	Forced elements to be processed at the same station for all models. Formulation of mixed model problem as single model.	Based on the assumption that each element requires the same amount of time for all models, but it may be omitted in some models. The equipment necessary for each element has to be present only at one station. Operators specialize only in a few tasks. Slight modification of sequencing procedure of Wester and Kilbridge.	The technique does not provide a mechanism for generating solutions. It can be used with any procedure, and its results are only as good as the procedure.

Table 2. (cont.)

Author(s)	Year	Article	Contribution	Characteristics	Shortcomings
3. Thomopoulos	1970	Mixed Model Line Balancing with Smoothed Station Assignments.	Objective function that minimizes the difference in work contents between stations as well as between models.	Balancing only. Minimization over the proposed objective function promotes better sequencing as well as high productivity for the given number of stations. No particular mechanism for generating solutions is specified.	Number of stations has to be known beforehand and any procedure might stall, looking for solutions with the ideal number of stations while there are not any.
4. Dar-El, Cucuy	1970	Optimal Mixed-Model Sequencing for Balanced Assembly Lines.	Exact method for sequencing. Objective function is to minimize product idle time with zero operator idle time.	Sequencing only. Formulation and algorithm based on Integer Programming. The procedure is for fixed rate launching.	The algorithm assumes near perfect line balancing, and in turn is unsuitable for problems with too many precedences and/or too few stations.

Table 3. Work on the Single Model Stochastic Problem

Author(s)	Year	Article	Contribution	Characteristics	Shortcomings
1. Moodie, Young	1965	A Heuristic Method of Assembly Line Balancing for Assumptions of Constant or Variable Work Element Times	First Work on Stochastic Task Times. Quick balances for the stochastic case, with smoothed station times.	Heuristic Algorithm. In a first phase, elements are distributed to stations, possibly in an infeasible balance, and in the second phase they are rearranged for a smoother balance.	The proposed solution is very dependent on the initial balance. A very small number of feasible solutions are investigated, and the proposed solution is not guaranteed to be any close to the optimum.
2. Kottas, Lau	1973	A Cost Oriented Approach to Stochastic Line Balancing	Incompletion Costs are considered. The proposed algorithm is fast and efficient in proposing feasible solutions.	Heuristic Algorithm. One station is balanced at a time. Elements are assigned on the basis of their incompletion costs. A procedure is used for station improvement before advancing to the next station.	Only normally distributed task times are considered. The algorithm is very inflexible, since after all stations are balanced, no further improvement is made. It searches only a small part of the region of feasible solutions.

Table 3. (cont.)

Author(s)	Year	Article	Contribution	Characteristics	Shortcomings
3. Sarin, Erel	1989	Development of a Cost Model for the Single-Model Stochastic Assembly Line Balancing Problem	Better estimation of incompletion probabilities for the cost model proposed by Kottas and Lau. A dynamic programming procedure also proposed, would yield optimal solutions when combined with the cost model, if memory and time restrictions did not apply.	Heuristic Algorithm. The cost model is based on total enumeration of possible sets of complete and incomplete tasks at each point, and accurate calculation of the probabilities of each set occurring. The transfer function for a dynamic procedure based on this model is also provided. The resulting procedure is used as a heuristic with the application of boundaries on the generated solutions.	The cost model is very complex. Even in its heuristic application, the dynamic programming procedure is computa-tionally intensive. It leads to near optimum solutions only if the restricting boundaries are very loose, in which case a lot of time is needed for realistic problems.

Kottas and **Lau** (1973) developed a cost model for the operation of an assembly line under the assumption that incomplete tasks are finished off the line. They proposed a heuristic for optimizing over the cost model and improved the heuristic twice (1976, 1981). Their work also considered only normally distributed times. Their heuristic is very inflexible and a very small part of the region of feasible solutions is investigated, since only one station is balanced at a time.

Sarin and **Erel** (1989, 1990) proposed an improvement of the way incompletion probabilities are calculated to fit into Kottas and Lau's cost model. Compared to the work of Kottas and Lau, Sarin and Erel's method provides with accurate cost estimates. It is based on total enumeration of possible sets of complete and incomplete tasks at each point, and on accurate calculation of the probabilities for each set to occur. In addition, any probability distribution can be utilized and two heuristics were introduced. One of the heuristics (1990) is based on dynamic programming and includes a transfer function for the cost model. It is computationally intensive, and the results can be inferior if too many restrictions are imposed to speed up the calculations. The other heuristic (1989) is a branch and bound procedure for finding approximate solutions to the problem.

2.1.4. Mixed Model Stochastic Problem

The very few relevant articles on the mixed model stochastic problem are summarized in Table 4.

Vrat and **Virani** (1976) proposed a method to reformulate the mixed model stochastic problem into a single model problem. To achieve this, they assumed element times to follow a probability distribution function for only one basic model, and they used the average distributions to

Table 4. Work on the Mixed Model Stochastic Problem

Author(s)	Year	Article	Contribution	Characteristics	Shortcomings
1. Vrat, Virani	1976	A Cost Model for Optimal Mix of Balanced Stochastic Assembly Line and the Modular Assembly System for a Customer Oriented Production System	Reformulation of the mixed model stochastic problem into a single model stochastic problem. Solution of the resulting problem with the procedure and cost model of Kottas and Lau.	Balancing only. All models are considered as variations of a single, basic model. Task times are considered to follow one distribution for each element, the same for all models. Sequencing is not necessary.	This approach may be used only where differences in task times between models are very small. Otherwise, with the lack of sequencing, random model sequences might give very poor results.
2. Okamura, Yamashina	1979	A Heuristic Algorithm for the Assembly Line Model-Mix Sequencing Problem to Minimize the Risk of Stopping the Conveyor	Reformulation of the sequencing problem with the objective of minimizing the probability of stopping the conveyor. An efficient heuristic algorithm was proposed, that yields results comparable to those of a branch and bound procedure.	Sequencing only. A basic assumption is that incompletions have to be dealt with on the spot. The objective was achieved by maximizing the least operator allowance for all stations and models. The heuristic algorithm improves an initial solution by means of a series of interchanges of models.	Only expected task times were considered. The variabilities of task times that really determine the probability of stopping the conveyor were not taken into account. The heuristic algorithm proposes solutions dependent on the initial solution. It can yield very good sequences only if the initial solution is good enough.

determine this single distribution for each model. With this kind of an approach, sequencing is unnecessary. They used this approach to solve a real problem from industry, with an extremely large number of models. After reformulating it, they used the procedure of Kottas and Lau to solve it. A necessary condition for this approach to be used is that the differences in task times among models are very small. In the opposite case, the lack of sequencing could lead to very poor solutions.

2.1.5. Computer Programs

There are a number of computer programs that facilitate the design of an assembly line. The most important ones are outlined in Table 5.

A computer program with wide applications is **COMSOAL**, developed by **Arcus** (1966). It can handle both single and mixed model lines, with either deterministic or stochastic times, and specializes in handling large problems. It applies heuristic rules in order to come up with a solution, and it uses a multi-pass procedure: it randomly generates a number of solutions and selects the best one among them. As a result, it is very fast at generating results, but the solutions are not guaranteed to be close to the optimum.

Another successful program was **CALB**, developed by **Kovach** (1969). It tackles the mixed model deterministic case, can handle a large variety of constraints, and is designed for production floor usage to solve problems of realistic size. The generated solutions are again not guaranteed to be close to the optimum.

Table 5. Computer Programs for Assembly Line Design

Developer	Year	Article	Features	Efficiency
1. Arcus	1966	COMSOAL: A Computer Method for Sequencing Operations for Assembly Lines	Single and Mixed model lines. Deterministic task times. Balancing only. Heuristic rules are combined with a multi-pass procedure that generates random solutions and selects the best one among them.	Fast results. Because of random generation of solutions, they may not be close to the optimum. Good for extremely large problems.
2. Kovach	1969	CALB: Systematic Assembly Management	Mixed model lines. Deterministic task times. Balancing and Sequencing. A large variety of possible types of constraints are included. Solutions are produced by use of heuristic rules.	Fast results. Solutions are not guaranteed to be close to the optimum. Very good at generating feasible balances and sequences for large problems in very little time. Very much oriented for use in the production floor.
3. Smith	1990	A Computerized Search Methodology for the Design of Mixed Model Assembly Systems	Mixed model lines. Deterministic task times. Exhaustive Balancing and Sequencing. Balancing can be performed stepwise, for a number of stations at a time, for faster results in the case of very large problems.	Fast results. Solutions very close to the optimum need a lot of time to be generated. The stepwise use of the program is an intermediate situation and makes the program flexible. Sequencing is optimum in any case.

Recently, **Smith** (1990) developed a computer program in Prolog, which exhaustively balances mixed model deterministic problems of moderate size. Its main advantage is that it provides with optimum solutions in very little time. For problems of large size, it includes the capability of balancing only a range of stations at a time, and thus comes up with balances close to the optimum in reasonable time.

2.2. Assembly Line Sequencing

2.2.1. Mixed Model Deterministic Problem

The most important work on the mixed model deterministic problem is summarized in Table 2. In the first work on the mixed model problem by **Wester** and **Kilbridge** (1964), detailed procedures for sequencing variable or fixed rate launching were described. The objective was to avoid both operator idle time and work congestion. The procedures are easy to apply, but are guaranteed to produce a balance that minimizes idle times only for one model at a time. **Thomopoulos** (1967) modified the procedure of Kilbridge and Wester for sequencing by assuming different labor costs for each group of stations.

Dar-El and **Cother** (1975) introduced a set of heuristics for sequencing that adds one model to the sequence at a time. All the sequencing methods mentioned so far are heuristics and do not guarantee that inefficiencies will disappear, or that idle time is minimized.

Dar-El and **Cucuy** (1977) proposed an exact method for sequencing, based on an integer programming formulation of a closed station assembly system. Their objective was to minimize the product idle time, while maintaining minimum operator idle time, for fixed rate launching. However,

one of their assumptions is that the line balance is almost optimal. Therefore, the algorithm is unsuitable for problems with too many precedences and/or too few stations.

2.2.2. Mixed Model Stochastic Problem

The major work on sequencing for the mixed model stochastic problem is summarized in Table 4. **Okamura** and **Yamashina** (1979) produced a heuristic for solving the sequencing problem, based on minimizing the probability of stopping the conveyor because of incompletion of a task within the cycle time. However, the probability distributions of elemental times are not used to determine the probability of stopping the conveyor. Instead, an attempt was made to maximize the least operator allowance, based only on expected task times. Another disadvantage of this approach is that the proposed procedure is dependent on an initial sequence and the proposed solutions are only partial modifications of the initial solution.

2.2.3. Computer Programs

Computer programs that address sequencing are outlined in Table 5. **CALB**, the program developed by **Kovach** (1969), included a feature for sequencing a mixed model line. No more details on sequencing by this program can be found in the literature.

Smith (1990) developed a sequencing program which exhaustively investigates all possible sequences for a given balance. With relatively little computing effort, the program evaluates all sequences and selects the best one.

CHAPTER 3: DEVELOPED METHODOLOGY

3.1. Assumptions

The developed methodology is based on the following assumptions:

Type of Problem: The problem addressed is the mixed model stochastic assembly line balancing and sequencing problem.

Type of Stations: All stations are assumed to be closed to the left and open to the right. The operators cannot cross the left station limit, but are allowed to work in the next station for a limited time. Enlarged and multiple stations are not allowed.

Elementary Tasks: The assembly operation has been divided into elementary tasks which cannot be further divided. Each elementary task is to be performed by one operator with the same tooling. Each model requires some or all of the elementary tasks for its assembly.

Precedence Relations: Any established precedence relationships between tasks cannot be overridden. To perform any task, it is absolutely necessary that all of its predecessors have been completed first.

Element Times: Element times can be deterministic or probabilistic. In the case of probabilistic elemental times, the type of disrtibution used may be normal or uniform. They are independent of each other as well as of the operator performing the task. Similar elements are allowed to have different probability distributions for different models.

Costs Involved: The cost of **maintaining the assembly line** is constant and does not change with the number of stations or the speed of the transfer mechanism. In turn, it is not taken into account for optimization. **Labor costs** are proportional to the number of workers (or stations) in the line. Each operator is compensated with a fixed salary, (same for all) per shift. **Overtime** is not allowed, except for finishing incomplete tasks. **Incompletion Costs** are the same for all models for a particular task, and do not depend on the remaining time to complete the task.

Line operation: The line does not stop for incomplete tasks. An incomplete task cannot be completed in one of the next stations. Instead, this task and all its followers in the precedence diagram are completed off the line.

Scheduled Quantities: The scheduled quantities for each model are deterministic and known.

Delivery Times: Scheduled delivery times are not taken into account. It is assumed that all scheduled models have ample time to be completed, and that the only relevant constraint is to meet productivity requirements for the scheduled demand.

Equipment Duplication: The cost of duplicating existing equipment is not considered.

3.2. The Balancing-Sequencing Methodology

A complete assembly line design consists of two major elements:

1. A line balance, an allocation of work elements to work stations, as well as the particular order in which the elements are processed at the stations.

2. A line sequence, the exact sequence that the models will have to follow when they are fed into the assembly line.

The developed methodology consists of two stages. In stage I, all feasible line balances and model sequences are generated using an exhaustive algorithm. In stage II, expected total costs are determined for each balance - sequence combination. The methodology is described in terms of four major steps:

1. **Preliminary data manipulations:** This step must be performed manually and includes: preparation of a combined precedence diagram, decision on distribution functions that the work elements follow, decision on the stations configuration, and decision on lower and upper limits for expected station times, and additional parameters pertinent to generating solutions. The above decisions are of qualitative nature, and in turn are not undertaken by the developed computer program.

2. **Generation of sequences:** This step is undertaken by one of three modules of the developed computer package. It involves identifying all feasible model sequences.

3. **Generation of balances:** The second module of the computer package is responsible for identification of all feasible allocations of elements into stations. Ordering of elements within a station is performed in step 4.

4. **Evaluation and selection of feasible solutions:** The third module of the computer package performs two main tasks before proposing a set of recommended solutions. The first task involves determining the orders of elements within stations that result in the least cost for each balance. This task is performed in this module instead of the balancing module to minimize on necessary disk

space. The second task involves estimating the total cost for each balance and sequence combination.

3.3. Preparation of Data

In addition to the basic data required, outlined in section 3.3.1, to solve the problem, the software user must decide on some parameters which influence the magnitude of the exhaustive search. These are explained in the following sections.

3.3.1. Basic Problem Data

The number of elemental work tasks.

The number of models.

The number of available stations.

The scheduled quantity for each model per shift.

The probability distribution function of the processing time for each element and model.

The desired production rate or, alternately, cycle time.

The duration of each shift.

The precedence relationships between elements.

Any zoning constraints.

The uniform labor rate.

The incompletion cost for each element.

The combined precedence diagram and the configuration of work stations influence the total number of feasible solutions and the total system cost. Their influences are explained below:

Combined Precedence Diagram

The combined precedence diagram introduced by **Thomopoulos** (1970) is a good tool for mixed model problems. Similar element tasks for different models are labeled with the same serial numbers, and then the individual precedence relationships for elements of each model are combined into a single diagram. By labeling similar tasks with the same number, they are all forced to be processed at the same station and in the same order for all models. This way, only one model has to be balanced, the one represented by the combined diagram. A very good reason to use this practice is to avoid the duplication of very expensive equipment. In the combined diagram, a precedence relationship that would hold only for some of the models, now holds for all of them. An example of individual and combined diagrams is illustrated in Figure 2(a).

This practice is not suitable for all kinds of problems. When similar elements have inverse precedence constraints for different models, the resulting combined precedence diagram would have to include contradictory constraints. A solution to this problem is to give separate labels to some or all of the elements with conflicting precedences. An example is illustrated in Figure 2(b).

Another problem with using combined precedence diagrams is that they make good feasible solutions more unlikely to exist. Not all elements require expensive equipment, nor do precedences have to hold for all models. By giving common labels to all similar elements, the problem is restricted too much and the area of feasible solutions becomes extremely small. On the other hand, by using individual precedence diagrams, the problem solving procedure becomes too complicated since there are too many nodes and arcs in all diagrams together. Also, a high risk of having to duplicate expensive equipment appears. A remedy to this conflict is to give common labels only to a number of

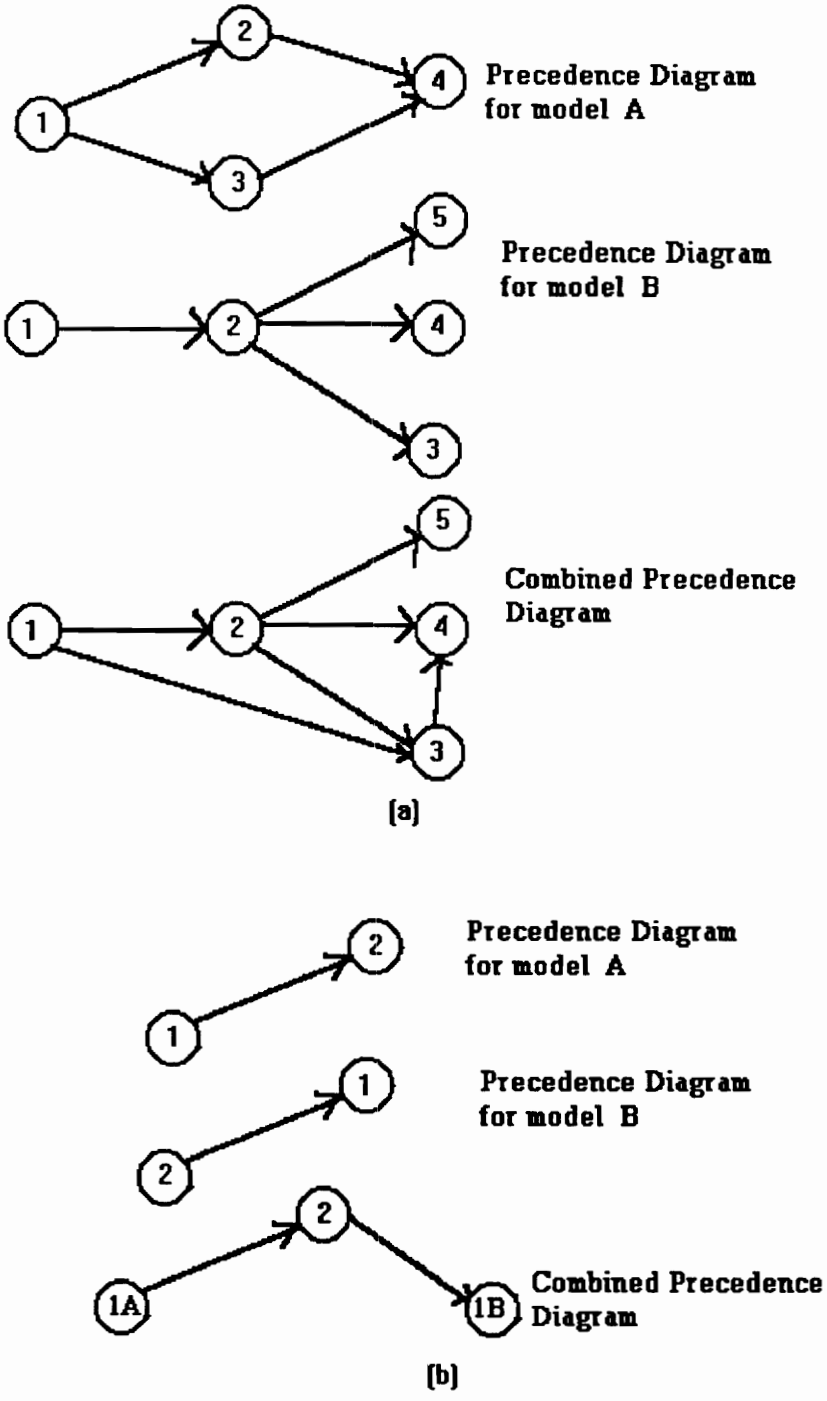


Figure 2. Combined Precedence Diagrams

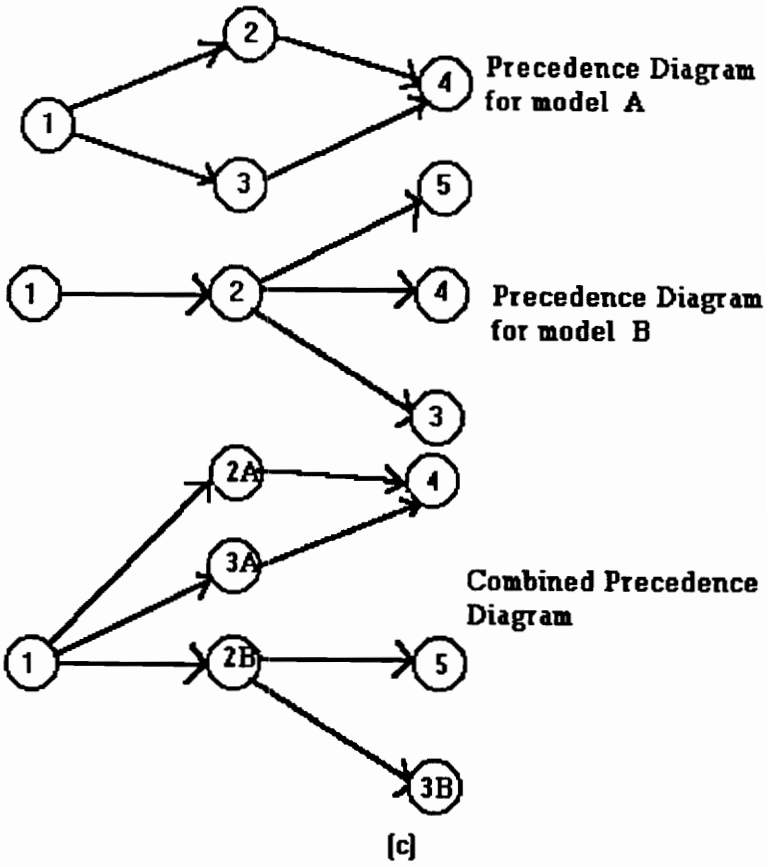


Figure 2. Combined Precedence Diagrams

similar tasks, giving priority to tasks with high cost of equipment and tools. An example where only tasks 1 and 4 require expensive tools is illustrated in Figure 2(c).

Configuration of Work Stations

The configuration of a typical work station is determined by two parameters. The fixed rate c used for launching, and the operator allowance a. Each station is closed to the left and open to the right. The parts are supposed to move from left to right, and it takes each of them c time units to move from the left to the right boundary of the station. The distance between boundaries is equal to $v\,c$, where v is the constant speed of the transfer mechanism. The operator is not allowed to cross the fixed boundary, but may continue work past the right imaginary boundary up to a time units after the part entered the station. The fixed left boundary of a station is the same as the imaginary right boundary of the previous station. A configuration of work stations is illustrated in Figure 3.

After a part has entered a station, the operator at this station may start work on the part only when two conditions have been met: (1) he has stopped work on the previous part, and (2) the operator at the previous station has stopped work on the current part. Work on a part stops when either all work assigned to the particular operator for the current model has finished, or the part has remained in the station for a time units and some of the assigned tasks remain incomplete.

The selection of values for c and a can greatly influence the performance of the assembly line. Since c is the rate at which the products are fed into the line, it is reasonable to use a c equal to the desired cycle time, which is dictated by the desired productivity rate. A smaller c would ensure an enormous amount of incompletions whereas a bigger one would make meeting the desired productivity impossible.

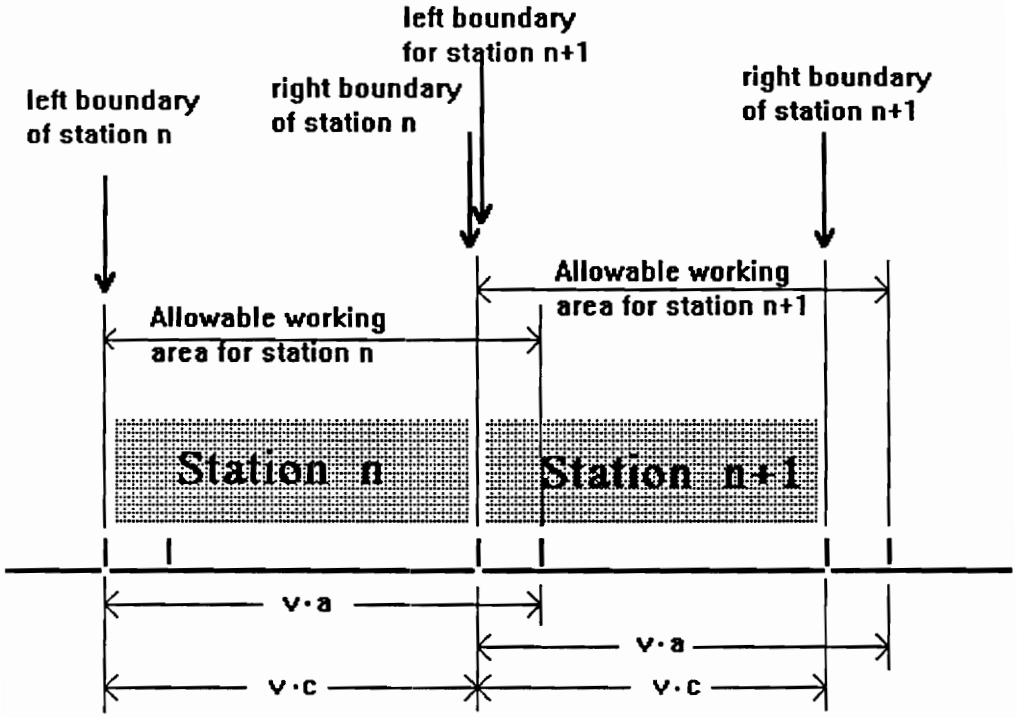


Figure 3. Configuration of Work Stations

To select a, one must consider the amount of variation of average total work contents among models. The role of a is that of a time buffer, for the operator to work more than average on models that require more service time than the average station time, using time he gained on others. Technological restrictions are the ones that limit the value of a, which has always to be greater than c. The bigger it is, the better it can handle times over the average work contents for some parts. If it is too small, the operator is not able to handle parts with more than average station time. If it is too big though, operators tend to always work on a delay, and parts with very small service times are also in danger of remaining incomplete.

3.3.2. Limiting Parameters for the Exhaustive Search

A number of parameters that influence the optimality and speed of the obtained solution have to be decided upon by the designer of the assembly line. These parameters can limit the scope of the search for feasible solutions to speed up the process of obtaining a solution. The computer programs can be directed to overlook solutions with too high or too low station service times, or with high variation of service time among stations and models.

The limiting parameters for the exhaustive search can be viewed in terms of two groups. One group addresses the size of the feasible region, such as an upper limit to station service time. The second group deals with the degree of exhaustive search for generating balances within the feasible region.

Feasible Region Size Parameters

By limiting the region of feasible solutions reasonably, one may obtain an optimal solution or one very close to optimum in a very short period of time. However, by limiting the region too much, there is the risk of excluding all good solutions or even all feasible solutions.

In general, problems with fewer precedence constraints would have more feasible balances. If the time to find a solution is limited, problems of this kind can be more safely restricted than problems with too many precedence constraints because of the higher probability of finding a good solution.

One good restriction is to limit the allowable number of stations. An upper limit may already exist because of available operators, floor space limitations, or technological constraints. If not, the line designer should select a maximum number of stations bigger than the theoretical minimum number dictated by the desired productivity. It is not necessary to impose any lower limit to the number of stations, since this is implied by the minimum number of stations where the tasks could ideally fit.

Another effective way to restrict the region of feasible solutions is to impose lower and upper limits on the average service time assigned to any individual station. A convenient upper limit is the operator allowable time a and should be imposed on each individual model. In practice, one would select an upper limit between c and a, since values too close to a would signify incompletions almost half of the time. For a lower limit, the line designer should select a value less than the average work assigned to the previously defined maximum number of stations.

Parameters affecting the Generation of Balances

Two sets of parameters influence the execution of the balancing module. The first set has two components, the look-up horizon and the maximum number of saved solutions. The program balances only a certain number of stations at a time as defined in the look-up horizon and saves only the desired maximum number of best solutions. In the following step, where the next set of stations are balanced, the program uses the saved solutions as initial balances.

The above two components are inter-related. For example, if only a few of the solutions are saved, a myopic strategy is followed, and the optimality of the selected solutions is not guaranteed unless the look-up horizon is increased. The longer the look-up horizon, the myopic phenomenon tends to disappear, since isolation and selection of partial best solutions is done in an expanded area. With a look-up horizon equal to the maximum number of stations, the saved solutions are guaranteed to be the best overall as an exhaustive search is used. However, it is advisable to set a smaller look-up horizon for problems of medium size, otherwise the computer programs will require an excessive amount of time (days for problems with 50 tasks and 20 stations) to generate balances.

The second set of parameters determines how many total solutions are to be generated at each look-up horizon and what is the maximum number of solutions to be generated for each of the saved solutions. For example, the total number of solutions to be generated at the look-up horizon can be set at 100,000, while the maximum number of solutions to be generated for each of the saved solutions of the previous look-up horizon can be set at 1,000. The maximum number of generated solutions at each step depends upon the size of the problem, the available time for its solution, and the established look-up horizon. The maximum number of solutions generated per saved solution may be used to ensure that at least some solutions are identified for every previously saved solution, before the number of

solutions generated per set of stations runs out.

3.4. Generation of Sequences

A computerized algorithm has been developed to generate all possible sequences for a schedule of models under some constraints similar to those of a precedence diagram. Although in the case of sequencing there are no real precedence constraints, yet they are imposed by the sequencing module to avoid duplicate sequences by creating arbitrary chains of identical products. An example of the resulting precedence diagram is given in Figure 4, where model A was chosen to be scheduled first.

In order to reduce the number of generated sequences, the model with the smallest scheduled quantity should be launched first. Also, instead of the actual scheduled quantities, their biggest common divisor (bcd) is found and the quantities divided by the biggest common divisor are used to produce the sequences. It is assumed that the best sequence will have a cyclic form, consisting of a number of repeated sequences.

The algoriithm used for generating sequences is shown in Figure 5, where an element stands for a model. In the algorithm, a list with a partial sequence of models is maintained. For every partial sequence, all elements that are not already a part of it are appended to it, as long as this is permitted by the precedence constraints. When all elements have been appended to a partial list, its last element is dropped and a new one replaces it. Whenever a new full sequence is obtained, it is stored in memory. The algorithm keeps modifying the partial list until all feasible sequences have been generated.

Model A: Scheduled Quantity 2
Model B: Scheduled Quantity 3
Model C: Scheduled Quantity 5

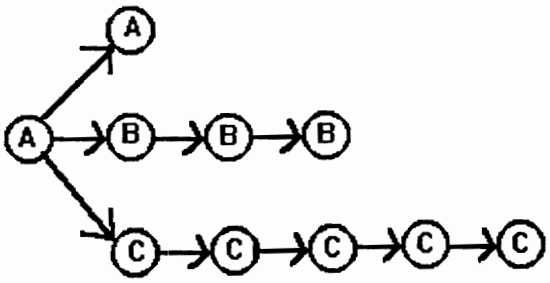


Figure 4. Model Sequencing in the form of a Precedence Diagram

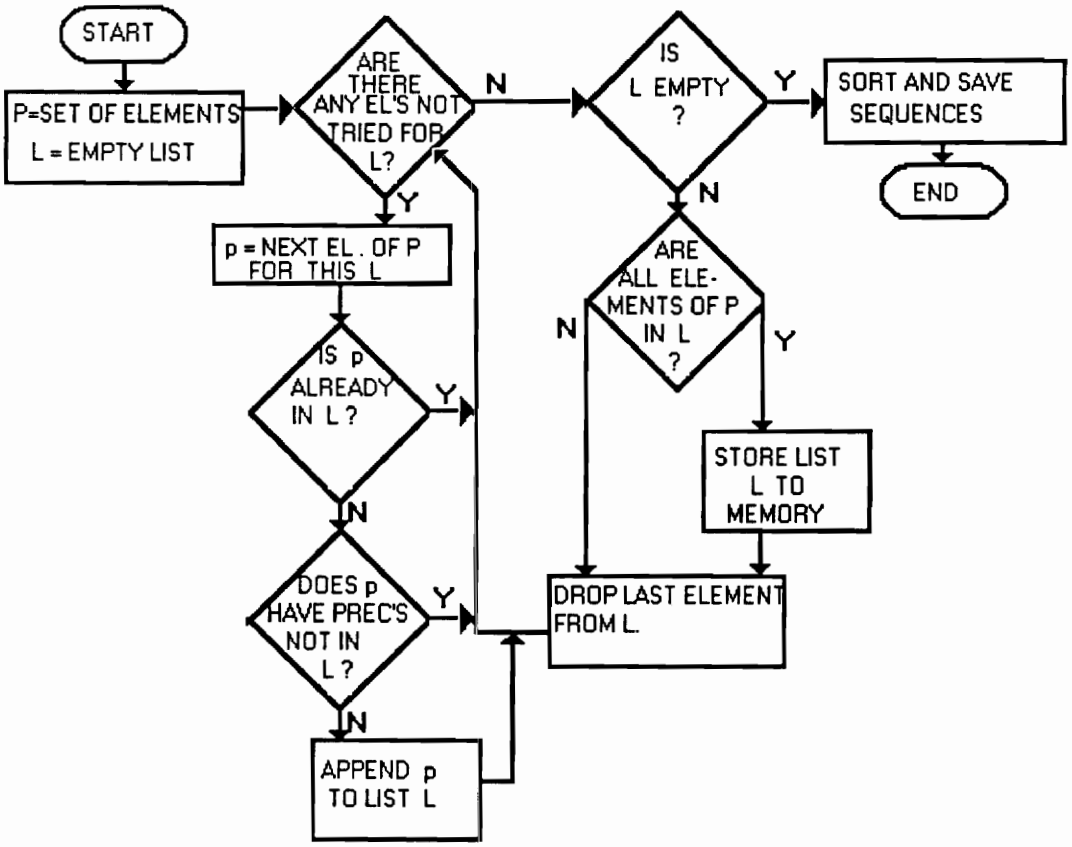


Figure 5. Algorithm for the Generation of Sequences

For each generated sequence, a performance measure reflecting product delays and extra operator times above the theoretical service time of a station is calculated. In general, if *twc* is the average total work content for the scheduled quantities and twc_i is the average total work content for the first i products in the sequence, then the performance measure for this sequence is

$$z = \sum_{i=1}^{q} \left| twc_i - twc \right|$$

where q is the total number of scheduled quantities.

For example, if we have two models with a total work content of 4 for model A and 5 for model B, and scheduled quantities of 3 for model A and 2 for model B, the performance measure will be calculated as follows:

For sequence A A B A B, twc = (3x4 + 2x5) / 5 = 4.4. twc_1 = 4, twc_2 = (4+4) / 2 = 4, twc_3 = (4+4+5) / 3 = 4.33, twc_4 = (4+4+5+4) / 4 = 4.25, twc_5 = (4+4+5+4+5) / 5 = 4.4, z = 0.5+0.5+0.167+0.25+0 = 1.417.

For sequence A B A A B, twc = (3x4 + 2x5) / 5 = 4.4. twc_1 = 4, twc_2 = (4+5) / 2 = 4.5, twc_3 = (4+5+4) / 3 = 4.33, twc_4 = (4+5+4+4) / 4 = 4.25, twc_5 = (4+5+4+4+5) / 5 = 4.4, z = 0.5+0.1+0.167+0.25+0 = 1.017.

The lower this measure, the more likely the sequence being part of a good solution. The sequences are sorted in ascending value of this measure, and then saved by the sequencing module in a file for subsequent use by the balance-sequence module.

3.5. Generation of Balances

The generation of balances algorithm is similar to the sequencing algorithm. The main difference is that whenever a new element is added to the list of the current station allocation, the station time is checked to determine whether it is possible to move to the next station. The generation of balances algorithm appears in Figure 6.

The algorithm provides for generating all sequences of work elements in the form of a list. Starting with an empty list, the algorithm keeps increasing it as the precedence constraints permit. Each time the station service time limits are satisfied, the algorithm advances from the current station to the next. The list is increased and decreased as necessary, to ascertain that all feasible sequences of elements are generated. Whenever it becomes full (containing all elements) the corresponding sequence is stored.

The basic restrictions to the generation of feasible solutions for faster results are precedence constraints and upper and lower station time limits. Also, there is a number of additional heuristic rules that have been implemented in the computer program to provide shortcuts for the algorithm due to the combinatorial nature of the problem. Without these heuristics, the generation of balances would require a lot of computational time. The utilized rules are as follows:

1. When exhaustively balancing, look ahead only for a limited number of stations. This rule has to be applied when the problem is too big to be solved exhaustively. It reduces the total necessary amount of time and computations dramatically, but does not guarantee optimal solutions. However, it is better than any type of random multi-pass procedure, since the results obtained are consistent and controllable instead of being randomly generated.

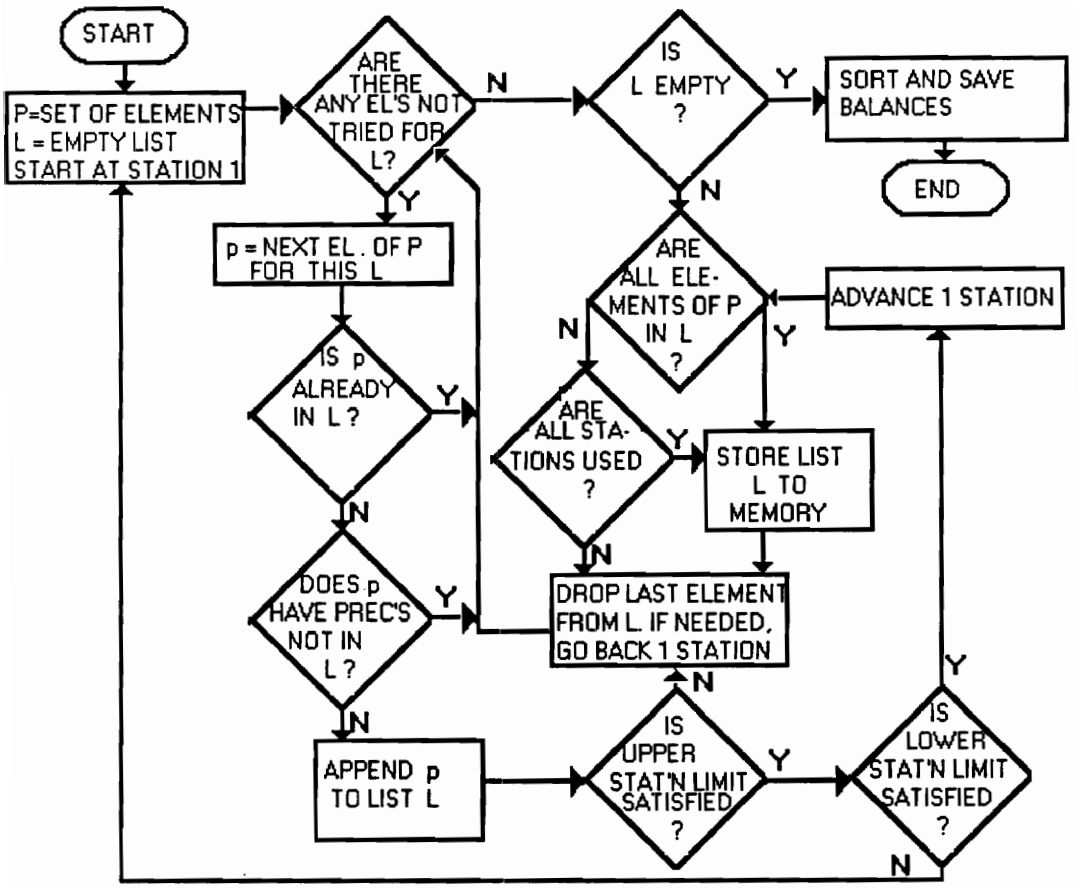


Figure 6. Algorithm for the Generation of Balances

2. Confine the allowable cycle time by an upper and lower limit. This eliminates any solutions with station times far away from the ideal cycle time. In any case, such solutions are not good, since they tend to require a large number of stations.

3. Do not continue balancing along a path when cumulative estimated costs become more than the estimated cost for the worst saved solution so far. The same practice is very useful in the final stage of sequencing also, where overall costs are estimated.

4. Do not consider rearrangements of the same set of tasks for the same station twice. Arrange tasks within stations looking only at their respective serial numbers. Ordering of tasks within stations is undertaken in the balance-sequence module.

5. Avoid permutations of the station orders. For a pair of stations with no interdependency, arrange the stations according to their smallest element labels.

Whenever one of these rules applies to a partial list, the list is disregarded and the program looks for other solutions that do not contain this list. Rules 3 and 5 do not affect the solution quality but only speed up the search. Other rules are expected to affect the optimality of the solutions, but to a very limited degree.

Of all the heuristic rules, the one that economizes on the time necessary to find all feasible solutions is the imposing of a look-up horizon (Rule 1). The objective of this technique is to split the balancing problem into subproblems of balancing a few stations. A number of solutions to each subproblem are saved, and are used as beginning balances for the next subproblem. The proper selection of maximum solutions generated per subproblem and per previously saved solution ensures

that the solutions generated and saved are representative and scattered evenly among all of the feasible region.

There is an upper limit to the number of solutions saved at each step. These are selected from the rest of the generated solutions based on the following performance measure:

$$\delta = \max_{i} \sum_{j=1}^{J} N_j \left| \frac{\sum_{k=1}^{K} \hat{t}_{jk}}{n} - p_{ij} \right|$$

where n is the number of stations in the current solution and i signifies a station. This measure is slightly different from the one defined by **Thomopoulos** (1970) who sums over i:

$$\sum_{i=1}^{n} \sum_{j=1}^{J} N_j \left| \frac{\sum_{k=1}^{K} \hat{t}_{jk}}{n} - p_{ij} \right|$$

Thomopoulos' measure was devised for the mixed model deterministic problem, where a value of zero indicates a perfect balance and sequence. For a perfect solution, the values of a and c can be set equal to the desired cycle time.

The same applies to the measure δ with the difference that the maximum instead of the total distance from a perfect solution is used. This is due to the stochastic times, where more incompletions are bound to occur in the station with the worst δ (in terms of difference from average assigned times per model and station), and this is the station monitored in the proposed measure. The solutions with minimum δ's are stored in a file in ascending order.

3.6. Evaluation of Balance - Sequence Combinations

The third module uses the solutions saved by the balancing and sequencing modules to form feasible sets of balances and sequences to evaluate their expected cost. This is done by retrieving balances one at a time and combining them with all stored schedules.

Before any combination takes place, the retrieved balance is first analyzed to determine the order of the work elements of each station for a given model. Next, all possible ordered balances are determined and evaluated in terms of expected incompletion cost. For each model, the ordered balance with the least expected incompletion cost is selected. For example, if there are two models, two ordered balances will be saved.

The expected total cost per shift including labor and incompletion costs is calculated by this module. The 50 combinations of sequence and balance with the least cost are presented to the user.

The combinations of balances and sequences are evaluated according to their total expected cost EC:

$$EC = bcd \cdot \sum_{k=1}^{K} \sum_{m=1}^{M} \sum_{q_i \subseteq Q_k^m} IC_{q_i}^{k,m} + K \cdot L \ .$$

The term K L is the necessary labor cost. K is the total number of stations in the current solution and L the uniform labor cost. The term bcd represents how many times the sequence in question is repeated per shift.

The set Q^m_k that is defined later contains elements assigned in station k. The term $IC^{k,m}_{qi}$ signifies the expected incompletion cost per reduced sequence incurred by incompletion of element q_i after it has started at station k, when part m is processed, for all possible combinations of previously finished and unfinished elements for this product. It includes incompletion costs for all additional elements that will have to remain incomplete because of q_i's incompletion. The method in which $IC^{k,m}_{qi}$ is calculated is explained later.

For each combination of a balance and a sequence, the cost of one product per station is examined at a time. For example, all products and their costs in station 1 are examined, according to their sequence. The same is repeated for all products in the next station, and so on.

At each step, the point of interest is the probabilities of the elements of the current product remaining incomplete given they have started in the current station. The problem is very complex, since a lot of non-trivial implications occur. The time the operator starts working on the product can lie between 0 and a - c time units after the part has entered the station based on the previous history of the product in the previous stations and of the operator with the previous products. Also, given that there is enough time, not all tasks are allowed to start, because some of their precedents may have

remained incomplete in previous stations.

At a particular product and station, let us assume that

$$Q^m_k = \{q_1, q_2, \ldots, q_l\}$$

is the set of elements that are allowed to begin, based on the m product's history at station k. This set (Q^m_k) is a subset of an ordered balance. Also, let R_{k-1} be the set of elements assigned to all previous stations and a subset of it:

$$r = \{r_1, r_2, \ldots, r_n\}$$

is the set of completed elements so far. The following must therefore be true :

- For every r_i in r, all its precedent elements (Pr_i) must also be included in r.

- All elements assigned in previous stations (R_{k-1}) that do not appear in r are incomplete. This can happen if a precedent of an incomplete element was incomplete in a previous station, or an incompletion occurred in the current station for the current element or another element earlier in the station's order.

These two properties are checked during the calculation of the probabilities of a given history to establish its validity. Once r is determined, it is possible to determine the elements of set Q^m_k. At this point, let us concentrate on calculating the probabilities of incompletions of $q_1, q_2, \ldots, q_l$. If we let d correspond to the delay caused by the history of the product and operator, and define

$F_{t_1, t_2, \ldots, t_i}(x)$ as the cumulative distribution function of the sum of times $t_1, t_2, \ldots, t_i$ (This is the convolution of $F(t_1)$, $F(t_2)$, ... , and $F(t_i)$), then the probability of q_1 starting within time a is given by:

$$F_d(a)$$

This probability is always equal to 1, since d cannot be bigger than $(a - c)$. The probability of q_1 starting but remaining incomplete is therefore:

$$F_d(a) \cdot \left[1 - F_{d,q_1}(a) \right].$$

The probability of q_2 starting is the same as the probability of q_1 finishing, which is

$$F_{d,q_1}(a).$$

Extending these probabilities, the probability of element q_i starting is

$$F_{d,q_1,q_2,\ldots,q_{i-1}}(a) .$$

The probability of it finishing is

$$F_{d,q_1,q_2,\ldots,q_i}(a) .$$

The probability of it remaining incomplete after starting is

$$F_{d,q_1,q_2,\ldots,q_{i-1}}(a) \cdot \left[1 - F_{d,q_1,q_2,\ldots,q_i}(a) \right].$$

These probabilities are dependent on the fact that r is the history of the part. The probability of element q_i remaining incomplete after it started is

$$\sum_{r \subseteq R_{k-1}} \text{Prob}(r) \cdot F_{d,q_1,q_2,\ldots,q_{i-1}}(a) \cdot \left[1 - F_{d,q_1,q_2,\ldots,q_i}(a) \right]$$

where $q_1, q_2, \ldots, q_n$ are elements of the Q^m_k that is dictated by r.

For each history r, the incompletion cost is increased by the incompletion probabilities of each element q_i in Q^m_k multiplied by the incompletion costs of all elements assigned in station k after q_i and all its followers that could be started given q_i was completed. The incompletion cost incurred by element q_i for history r is thus:

$$\text{Prob}(r) \cdot F_{d,q_1,q_2,\ldots,q_{i-1}}(a) \cdot \left[1 - F_{d,q_1,q_2,\ldots,q_i}(a) \right] \cdot \sum_{j \subseteq D_i} I_j$$

where D_i is the set of elements that are assigned after element q_i in station k and its followers that could be started given q_i was complete for history r.

The total incompletion cost for element q_i is then:

$$IC_{q_i} = \sum_{r \subseteq R_{k-1}} \text{Prob}(r) \cdot F_{d,q_1,q_2,\ldots,q_{i-1}}(a) \cdot \left[1 - F_{d,q_1,q_2,\ldots,q_i}(a) \right] \cdot \sum_{j \subseteq D_i} I_j$$

Prob (r) is equal to 0 for r's that are not valid, based on the previous two properties. Prob (r) is calculated as the product of Prob (r_k) for all k's, where r_k is the set of all elements of r assigned to station k. If $r_k = \{r_1, r_2, ..., r_l\}$, then Prob ($r_k$) may be defined in two ways:

$$\text{Prob}\ (r_k) = F_{r_1, r_2, \ldots, r_l}(a)$$

if there are no elements assigned to station k after r_l that could have started, and

$$\text{Prob}\ (r_k) = F_{r_1, r_2, \ldots, r_l}(a) \cdot \left[1 - F_{r_1, r_2, \ldots, r_{l+1}}(a) \right]$$

where r_{l+1} is the first element assigned in station k after r_l that could have started.

The delay d is dependent on the history r and the history of the operator. It is the biggest of d_1 and d_2, where d_1 is the operator delay and d_2 is the product delay. Both d_1 and d_2 lie within 0 and a - c. The way delays are inherited is illustrated in Figure 7. When an operator processes a part, in order to calculate the delay d' that will be carried to the next operator and/or station we need to find probabilities for three instances:

- All tasks are completed within c. The probability of this happening is $F_{d, q_1, q_2, ..., q_{i-1}}(c)$. In this case, d' = 0.

- All tasks are completed within a but not within c. The probability of this is $F_{d, q_1, q_2, ..., q_{i-1}}(a) - F_{d, q_1, q_2, ..., q_{i-1}}(c)$. In this case, $d' = d + q_1 + q_2 + ... + q_n - c$.

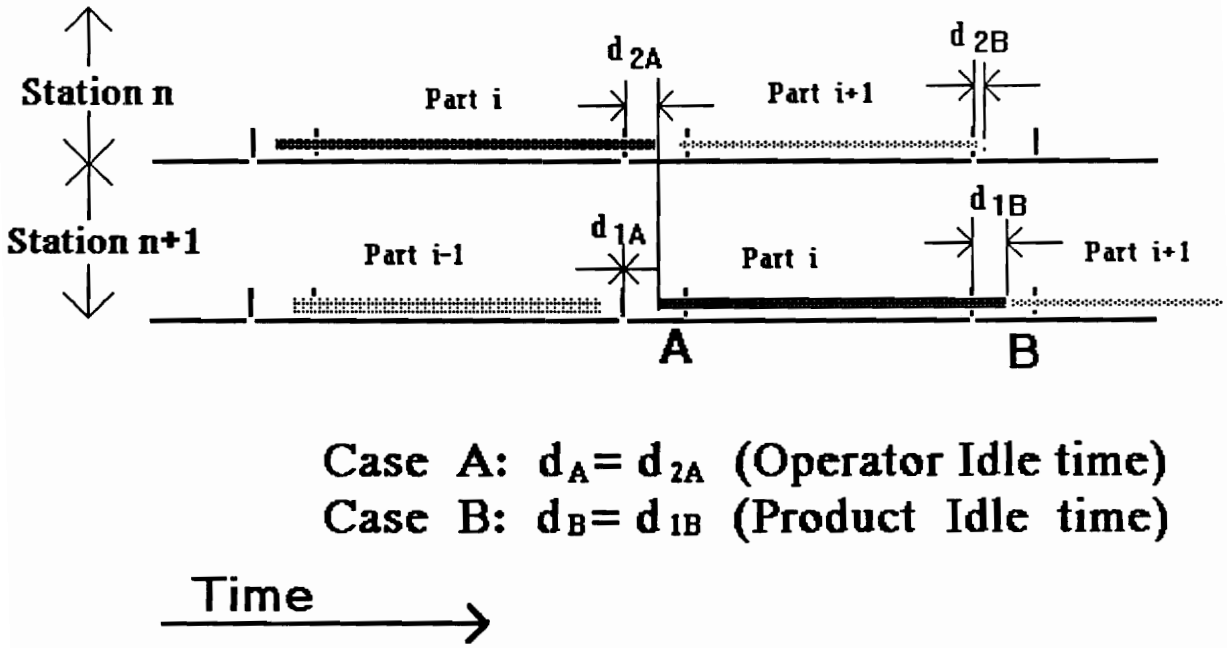


Figure 7. Inheritance of Delays by Previous Stations and Parts

- Incompletions occur, i.e. there are tasks that are not completed within a. The probability is $1\text{-}F_{d,q1,q2,...,qi-1}(a)$. In this case, $d' = a - c$.

Among d_1 and d_2, the one with a maximum value should be transfered to the current station and product. In the way that the module advances through stations first, and products second, the history is always known and d_1 and d_2 are also known. In the case where $d, q_1, q_2, ..., q_{i-1}$ all follow the normal distribution or i is big enough ($i \geq 10$), $d + q_1 + q_2 + ... + q_{i-1}$ also follows the normal distribution. Then, d' also follows the normal distribution between 0 and $a - c$ with known mean and standard deviation. In a new station, this is also true for d_1 and d_2. If we ignore the lower and upper limits for d momentarily and assume that both d_1 and d_2 follow unbounded normal distributions, then their maximum value $d = \max(d_1, d_2)$ also follows a normal distribution. If μ, μ_1 and μ_2 are the means and σ, σ_1 and σ_2 the standard deviations of d, d_1 and d_2 respectively, then

$$\mu = \mu_1 \Phi\left[\frac{\mu_1 - \mu_2}{\sqrt{\sigma_1^2 + \sigma_2^2}}\right] + \mu_2 \Phi\left[\frac{\mu_2 - \mu_1}{\sqrt{\sigma_1^2 + \sigma_2^2}}\right] + \sqrt{\sigma_1^2+\sigma_2^2}\sqrt{2\pi}\, e^{-\frac{(\mu_1-\mu_2)^2}{2(\sigma_1^2+\sigma_2^2)}}$$

$$\sigma = (\mu_1^2+\sigma_1^2)\Phi\left[\frac{\mu_1-\mu_2}{\sqrt{\sigma_1^2+\sigma_2^2}}\right] + (\mu_2^2+\sigma_2^2)\Phi\left[\frac{\mu_2-\mu_1}{\sqrt{\sigma_1^2+\sigma_2^2}}\right] + (\mu_1-\mu_2)\sqrt{\sigma_1^2+\sigma_2^2}\sqrt{2\pi}\, e^{-\frac{(\mu_1-\mu_2)^2}{2(\sigma_1^2+\sigma_2^2)}}$$

as obtained by Clark (1961) for a correlation of 0 between d_1 and d_2. (Φ (x) is the area to the left of x under the curve of the standardized normal distribution) Because of the limits imposed on d_1 and d_2, d also is limited by the same limits. This means that the values of the probability distribution function of d at 0 and $a - c$ are much higher than they should be with the normal distribution. Hence, d follows an unusual distribution function, with high probability of occurrence in the extremes. It is a truncated near-normal distribution, especially if all element times are normal, with the added weight of the truncated areas in the truncated limits. It could be estimated in three parts, with the 3 distinct

instances described above as obtained by the new distribution function of the maximum of d_1 and d_2. However, this would increase dramatically the complexity of the computational solution procedure. Instead, an approximation is used, considering d to follow a normal distribution with an average equal to μ and a standard deviation of

$$\frac{\min (a - c - \mu , \mu)}{3}$$

so that almost all of its values will range within 0 and a - c.

The total expected incompletion cost incurred for a complete sequence of products is

$$\sum_{k=1}^{K} \sum_{m=1}^{M} \sum_{q_i \subseteq Q_k^m} IC_{q_i}^{k,m}$$

Since the total number of products M is the actual number of scheduled quantities divided by their biggest common divisor bcd, then the total expected incompletion cost is

$$bcd \cdot \sum_{k=1}^{K} \sum_{m=1}^{M} \sum_{q_i \subseteq Q_k^m} IC_{q_i}^{k,m}$$

The cost model presented above for the calculation of incompletion costs is the same as proposed by Sarin and Erel (1989). The proposed procedure for calculating incompletion probabilities can be viewed as an extension of their work to accommodate for the mixed model stochastic case, although an entirely different methodology is utilized.

CHAPTER 4: ANALYSIS OF RESULTS

4.1. Validation of Computer Package

All parts of the developed computer package were tested and validated by solving a number of small problems that were manually solved. Intermediate results on all calculated quantities were obtained, and agreed with the respective manual results.

A number of bigger, closer to realistic size, problems were analyzed with the software to test its integrity, ease of use and its performance. These problems were analyzed several times with different parameters to test the amount of influence of the parameters on the obtained results. Two of these problems are described in the next section and their experimental results are discussed to highlight the merits of the developed methodology and computer package.

4.2. Example Problems

Of the few papers on the mixed model stochastic problem, none was found to present a particular case study with experimental results. Therefore, a mixed model deterministic problem from Thomopoulos (1970) was selected, and arbitrary data on the distribution types, variability of task times, and labor and incompletion costs were added. Only 8 tasks of the 19 in the original problem were used. The resulting problem appears in Table 6. The 10 best sequences according to the performance measure z used are presented in Table 7. The 10 best balances according to the measure δ are shown in Table 8.

Table 6. Data for Example Problem

Number of Elements: 8 Number of models: 3

Element	Precedent Elements
1	-
2	-
3	1
4	1, 2
5	2
6	2
7	4, 5
8	6, 7

Elemental Task Distributions

	Model 1			Model 2			Model 3		
El.	Dist.	Avg.	STD	Dist.	Avg.	STD	Dist.	Avg.	STD
1	norm.	1	0.1	-	-	-	norm.	1	0.1
2	cons.	0.4	-	cons.	0.8	-	norm.	1.2	0.1
3	cons.	0.2	-	-	-	-	-	-	-
4	norm.	0.5	0.05	norm.	0.5	0.05	norm.	0.5	0.05
5	-	-	-	norm.	0.5	0.05	norm.	0.55	0.05
6	cons.	0.4	-	cons.	0.3	-	cons.	0.2	-
7	cons.	0.1	-	cons.	0.3	-	cons.	0.5	-
8	norm.	0.7	0.1	norm.	0.1	0.1	norm.	1.5	0.2

Scheduled Quantities

Model 1	Model 2	Model 3
120	60	40

Labor Cost: 10

Element:	1	2	3	4	5	6	7	8
Incompletion Cost:	0.4	0.2	0.1	0.2	0.2	0.1	0.1	0.3

Table 7. Best Sequences for Example Problem

Sequence	1:	3	1	1	2	2	3	2	1	1	1	1	z = 3.7456
Sequence	2:	3	1	2	1	2	3	1	2	1	1	1	z = 3.7646
Sequence	3:	3	1	2	1	2	3	2	1	1	1	1	z = 3.7789
Sequence	4:	3	1	2	2	1	3	1	2	1	1	1	z = 3.7896
Sequence	5:	3	1	2	2	1	3	2	1	1	1	1	z = 3.8039
Sequence	6:	3	2	1	1	2	3	1	2	1	1	1	z = 3.8147
Sequence	7:	3	2	1	2	1	3	1	1	2	1	1	z = 3.8271
Sequence	8:	3	2	1	1	2	3	2	1	1	1	1	z = 3.8289
Sequence	9:	3	2	1	2	1	3	1	2	1	1	1	z = 3.8396
Sequence	10:	3	2	1	2	1	3	2	1	1	1	1	z = 3.8539

Table 8. Best Balances for Example Problem

The 10 best balances out of 68 investigated in terms of lowest Delta are :

Solution 1
Station 1 : 2 6 Time : 218.00 Dif : 54.67 (D = 54.67)
Station 2 : 1 4 5 Time : 322.00 Dif : 49.33 (D = 65.33)
Station 3 : 3 7 8 Time : 278.00 Dif : 5.33 (D = 29.33)
With a maximum Delta value of 65.33 and a total difference of 109.33

Solution 2
Station 1 : 1 2 Time : 304.00 Dif : 31.33 (D = 71.33)
Station 2 : 4 5 6 Time : 236.00 Dif : 36.67 (D = 56.67)
Station 3 : 3 7 8 Time : 278.00 Dif : 5.33 (D = 29.33)
With a maximum Delta value of 71.33 and a total difference of 73.33

Solution 3
Station 1 : 1 2 Time : 304.00 Dif : 31.33 (D = 71.33)
Station 2 : 4 5 6 7 Time : 286.00 Dif : 13.33 (D = 42.67)
Station 3 : 3 8 Time : 228.00 Dif : 44.67 (D = 44.67)
With a maximum Delta value of 71.33 and a total difference of 89.33

Solution 4
Station 1 : 1 2 Time : 304.00 Dif : 31.33 (D = 71.33)
Station 2 : 3 4 5 6 Time : 260.00 Dif : 12.67 (D = 32.67)
Station 3 : 7 8 Time : 254.00 Dif : 18.67 (D = 53.33)
With a maximum Delta value of 71.33 and a total difference of 62.67

Solution 5
Station 1 : 1 2 Time : 304.00 Dif : 31.33 (D = 71.33)
Station 2 : 4 5 7 Time : 212.00 Dif : 60.67 (D = 80.67)
Station 3 : 3 6 8 Time : 302.00 Dif : 29.33 (D = 38.67)
With a maximum Delta value of 80.67 and a total difference of 121.33

Table 8. (cont.)

Solution 6
Station 1 : 1 2 5 Time : 356.00 Dif : 83.33 (D = 83.33)
Station 2 : 3 4 6 Time : 208.00 Dif : 64.67 (D = 64.67)
Station 3 : 7 8 Time : 254.00 Dif : 18.67 (D = 53.33)
With a maximum Delta value of 83.33 and a total difference of 166.67

Solution 7
Station 1 : 1 2 Time : 304.00 Dif : 31.33 (D = 71.33)
Station 2 : 3 4 5 Time : 186.00 Dif : 86.67 (D = 86.67)
Station 3 : 6 7 8 Time : 328.00 Dif : 55.33 (D = 55.33)
With a maximum Delta value of 86.67 and a total difference of 173.33

Solution 8
Station 1 : 1 2 5 Time : 356.00 Dif : 83.33 (D = 83.33)
Station 2 : 3 4 7 Time : 184.00 Dif : 88.67 (D = 88.67)
Station 3 : 6 8 Time : 278.00 Dif : 5.33 (D = 14.67)
With a maximum Delta value of 88.67 and a total difference of 177.33

Solution 9
Station 1 : 1 2 Time : 304.00 Dif : 31.33 (D = 71.33)
Station 2 : 4 6 Time : 184.00 Dif : 88.67 (D = 88.67)
Station 3 : 3 5 7 8 Time : 330.00 Dif : 57.33 (D = 81.33)
With a maximum Delta value of 88.67 and a total difference of 177.33

Solution 10
Station 1 : 2 6 Time : 218.00 Dif : 54.67 (D = 54.67)
Station 2 : 1 5 Time : 212.00 Dif : 60.67 (D = 60.67)
Station 3 : 3 4 7 Time : 184.00 Dif : 88.67 (D = 88.67)
Station 4 : 8 Time : 204.00 Dif : 68.67 (D = 68.67)
With a maximum Delta value of 88.67 and a total difference of 272.67

(difference signifies the absolute difference from the ideal station time for 3 stations)

No heuristic rules were used for the generation of balances (the search was exhaustive). The only limits imposed were upper and lower limits that were 40 % above and below the ideal station time for a line with 3 stations. The 10 best balances and the 10 best sequences were used to find the combinations of balances and sequences with the least expected cost. The format of a combination of balance and sequence as presented by the program is shown on Table 9.

Two sets of combinations were produced: one for an operator allowance of a = 2.3 and another one for an operator allowance of a = 2.0. The cycle time for the 3 stations is 1.2934. All elemental times are in minutes. All combinations and their respective costs are summarized in Table 10 for a = 2.3 and in Table 11 for a = 2.0. The summarized results show multiple solutions with least costs.

4.3. Comparison with Previous Approaches

Traditionally, in the mixed model case, only one balance is selected based on some performance measure and a number of sequences are evaluated for this particular balance. Based on δ, the balance #1 of these presented in Table 8 would have been selected. Note that based on Thomopoulos' measure, balance #1 would have been selected. However, balances #2, 4, 5, 7 and 9 have better costs in the case where a = 2.3. Furthermore, use of Thomopoulos' measure would make balances #6 and 8 more preferable than 5, 7, 9 and 10, and would lead to significantly higher costs for both selected values of a.

Table 9. Form of a Complete Solution

Solution 14

Model 1:
Station 1: 1 2
Station 2: 4 5 6
Station 3: 3 7 8

Model 2:
Station 1: 2 1
Station 2: 4 5 6
Station 3: 3 7 8

Model 3:
Station 1: 2 1
Station 2: 4 5 6
Station 3: 3 7 8

With a maximum Delta value of 71.33
Sequence of models: 3 2 1 1 2 3 2 1 1 1 1 (repeated 20 times)

Total Expected Cost 35.920799

Table 10. Summary of Results ($a = 2.3$)

Balance	1	2	3	4	5	6	7	8	9	10
δ	65.33	71.33	71.33	71.33	80.67	83.33	86.67	88.67	88.67	88.67
$\Sigma\delta$	153.33	157.67	158.67	158.67	190.67	166.67	213.33	186.67	241.33	272.67

Sequence	z	Expected Cost of Combination									
1	3.8539	38.13	36.03	37.90	36.03	36.03	56.00	36.03	56.00	36.03	40.00
2	3.8039	38.26	36.03	37.85	36.03	36.03	59.10	36.03	59.10	36.03	40.00
3	3.8396	38.13	36.03	37.90	36.03	36.03	59.10	36.03	59.10	36.03	40.00
4	3.8289	36.51	35.92	37.81	35.92	35.92	56.00	35.92	56.00	35.92	40.00
5	3.7789	36.64	35.92	35.97	35.92	35.92	59.10	35.92	59.10	35.92	40.00
6	3.7896	38.26	36.03	37.85	36.03	36.03	62.21	36.03	62.21	36.03	40.00
7	3.8147	36.51	35.92	37.81	35.92	35.92	59.10	35.92	59.10	35.92	40.00
8	3.8271	38.27	36.03	37.90	36.03	36.03	59.15	36.03	59.15	36.03	40.00
9	3.7456	36.78	35.92	37.76	35.92	35.92	53.54	35.92	53.54	35.92	40.00
10	3.7646	36.64	35.92	35.97	35.92	35.92	62.21	35.92	62.21	35.92	40.00

Table 11. Summary of Results ($a = 2.0$)

	Balance	1	2	3	4	5	6	7	8	9	10
	δ	65.33	71.33	71.33	71.33	80.67	83.33	86.67	88.67	88.67	88.67
	$\Sigma\delta$	153.33	157.67	158.67	158.67	190.67	166.67	213.33	186.67	241.33	272.67
Sequence	z	Expected Cost of Combination									
1	3.8539	40.51	42.00	42.89	42.00	42.00	56.75	42.00	56.75	42.00	40.07
2	3.8039	42.69	42.00	42.88	42.00	42.00	64.01	42.00	64.01	42.00	40.07
3	3.8396	40.56	42.00	42.88	42.00	42.00	63.65	42.00	63.65	42.00	40.07
4	3.8289	40.45	42.00	42.88	42.00	42.00	57.06	42.00	57.06	42.00	40.07
5	3.7789	42.58	42.00	42.87	42.00	42.00	64.01	42.00	64.01	42.00	40.07
6	3.7896	42.72	42.00	42.87	42.00	42.00	70.91	42.00	70.91	42.00	40.07
7	3.8147	38.35	42.00	42.87	42.00	42.00	63.96	42.00	63.96	42.00	40.07
8	3.8271	44.40	42.00	42.88	42.00	42.00	68.08	42.00	68.08	42.00	40.07
9	3.7456	46.48	42.00	46.62	42.00	42.00	68.75	42.00	68.75	42.00	40.07
10	3.7646	40.48	42.00	42.86	42.00	42.00	70.91	42.00	70.91	42.00	40.07

All mixed model methods proposed so far, either select an arbitrary balance and find the best sequence for this balance according to some measure (Wester, Kilbridge, 1964, Dar-El, Cucuy, 1977, Okamura, Yamashina, 1979, Smith, 1990), or find a balance with good characteristics and assume that any random sequence will perform well with it (Thomopoulos, 1970, Vrat, Virani, 1976). However, as it is evident from the experimental results, no single balance or sequence may be guaranteed to give good results if it is not combined with the appropriate sequence or balance, respectively. Therefore, the technique of evaluating combinations of balances-sequences without presetting any of them proves very useful and can lead to significant reductions in the operation cost of an assembly line.

Another important contribution of the developed methodology is the development of a cost model for the mixed model stochastic problem and of a recursive procedure for calculating incompletion probabilities for the cost model. Previous works either addressed only the mixed model case (Kottas, Lau, 1973, Sarin, Erel, 1989) or were based on the assumption that the line stops each time an incompletion occurs (Okamura, Yamashina, 1979). The latter assumption is a bad industrial practice, since the whole assembly line is delayed, all of the operators have to remain idle, and the delivery of scheduled quantities is not guaranteed.

The proposed methodology guarantees accurate cost measurements and helps the line designer select the combination of a balance and sequence with the least cost. Its main drawback is that it requires a tremendous amount of time to produce final results in the exhaustive case. The generation of sequences takes only a few minutes, the generation of balances several hours for a problem of 50 elements and 20 stations with a look-up horizon of 3 stations, and the cost evaluation of combinations of balances and sequences close to one hour for a problem with 8 elements, close to 200 hours for 20 elements.

As the problem grows in terms of elements and stations, the execution time of the modules

grows almost exponentially. The valid histories for the cost evaluating module are selected from two to the Kth power, where K is the number of elements. However, the heuristic rules developed do speed up the balancing module and help in producing an adequate number of very good balances in very little time, compared to balancing without any heuristics.

4.4. Technical Characteristics of the Computer Package

The developed computer package consists of three modules that were developed in the C programming language. The Turbo C v.1.0 compiler was used to turn the code produced into executable files for the IBM PC/XT/AT and compatible machines. The source code can be found in Appendix A of this thesis, and is written based on the principles of structured programming. It is documented with extensive comments as part of it. All execution times mentioned in the thesis have been obtained on a 286 machine with a 16 Mhz processor and a math coprocessor.

The hardware requirements for the executable modules to run are (1) 256K of available RAM memory, and (2) enough free space on the current drive for reports and temporary files generated by the package to be saved. Their size depends on the size of the problem. A hard drive is strongly recommended.

CHAPTER 5: CONCLUSIONS

A computerized methodology was developed for the design of mixed model stochastic assembly lines. A computer package was developed to assist the assembly line designer in implementing the methodology and it was validated with an example from the literature.

No other published methodology was found to address the mixed model stochastic problem based on the utilized assumptions. The major assumption for this research work is that any products with unfinished elements remain on the line till they exit the last station for their incomplete tasks to be completed. With this approach, the line never stops and operators do not have to remain idle while they wait for the unfinished elements to be completed.

The first and second modules of the developed computer package generate feasible sequences and balances. Decisions on which balance and sequence to use are handled by the third module, where all combinations of the previously generated balances and sequences are identified, and their respective expected costs are calculated.

There are two main advantages in this treatment:

1. Each of the modules can run independently and the designer may repeat some of the steps with different parameters. This is particularly useful for the balancing module, since the proper selection of parameters determines whether there are any and how many feasible balances are generated.

2. Without analyzing both balances and schedules, the overall best combination cannot be obtained by any other known method.

When large, expensive lines are being designed, even small savings per shift can add up and justify the extra cost and effort to obtain an optimal combination. For cases where the necessary resources are not available, or where the line has to be reconfigured too often, the heuristic rules perform well enough and the solutions obtained are not in any case too far from the optimum. The advantage over other heuristic procedures specifically for large problems (Tonge, 1961, Arcus, 1966, Dar-El, 1973) is that the search is not limited in a random fashion, but in a very controlled way, scanning all of the feasible region for solutions.

The following areas may be investigated for future work:

1. Heuristic rules for finding approximate total expected costs. Only a number of elements may contribute to incompletion costs for each station, since the probabilities of elements assigned early in the stations remaining incomplete are extremely low.

2. More flexibility in the design of the assembly line. A computer program might help the line designer add multiple operators, enriched stations, and additional features to enhance the line productivity. Even the possibility of buffers within the line to enable each operator to work at his own pace might be considered.

3. The possibility of changing the line operation and the operator assignments while the line is operating. For instance, a task that will remain unfinished with 100% certainty should never start on the line, and the operator will know this will happen as soon as he finishes the previous task of the same part.

REFERENCES

Arcus, A. L., 1966, COMSOAL : A Computer Method for sequencing Operations for Assembly Lines. *International Journal of Production Research*, v. 4 n. 4, pp.259-277.

Betts, J., and **Mahmoud, K. I.**, 1989, Method for Assembly Line Balancing. *Engineering Costs and Production Economics*, v. 18 n. 1, Oct, pp.55-64.

Betts, J., and **Mahmoud, K. I.**, 1989, Identifying Multiple Solutions for Assembly Line Balancing having Stochastic Task Times. *Computers and Industrial Engineering*, v. 16 n. 3, pp.427-445.

Bowman, E. H., 1960, Assembly-Line Balancing by Linear Programming. *Operations Research*, v. 8 n. 3, May-June, pp.385-390.

Clark, E. C., 1961, The greatest of a finite set of random variables. *Operations Research*, v. 9 n. 2, March-April, pp.145-162.

Dar-El, E. M., 1973, MALB - A heuristic technique for balancing large single-model assembly lines. *AIIE Transactions*, v. 5, n. 4, Dec, pp.343-356.

Dar-El, E. M., and **Cother, R. F.**, 1975, Assembly Line Sequencing for Model Mix. *International Journal of Production Research*, v. 13 n. 5, pp.463-477.

Dar-El, E. M., Cucuy, S., 1977, Optimal Mixed-Model Sequencing for Balanced Assembly Lines. *OMEGA, the International Journal of Management Science*, v. 5 n. 3, pp.333-342.

Ghosh, S., and **Gagnon, R. J.**, 1989, Comprehensive Literature Review and Analysis of the Design, Balancing and Scheduling of Assembly Systems. *International Journal of Production Research*, v. 27 n. 4, Apr, pp.637-670.

Held, M., Karp, R. M., Sharesian, R., 1963, Assembly-Line Balancing-Dynamic Programming with Precedence Constraints. *Operations Research*, v. 11 n. 3, May-June, pp.442-459.

Helgeson, W. B., Birnie, D. P., 1961, Assembly Line Balancing using Ranked Positional Weight Technique. *Journal of Industrial Engineering*, v. 12 n. 6, Nov-Dec, pp.394-398.

Johnson, R. V., 1983, A Branch and Bound Algorithm for Assembly Line Balancing Problems with Formulation Irregularities. *Management Science*, v. 29 n. 11, Nov, pp.1309-1324.

Kalpakjian, 1989, Manufacturing Engineering and Technology, Addison Wesley, pp.22, 1115.

Kilbridge, M. D., Wester, L., 1961, Heuristic Method of Assembly Line Balancing. *Journal of Industrial Engineering*, v. 12 n. 4, July-Aug 1961, pp.292-298.

Kilbridge, M. D., Wester, L., 1961, The Assembly Line Problem. *2nd Operations Research Conference*, Aix-en-Provence, 1961.

Klein, M., 1963, On Assembly Line Balancing. *Operations Research*, v. 11 n. 2, Mar-Apr, pp.274-281.

Kottas, J. F., and **Lau, H.-S.,** 1973, Cost-Oriented Approach to Stochastic Line Balancing. *AIIE Transactions*, v. 5 n. 2, June, pp.164-171.

Kovach, J., 1969, CALB: Systematic Assembly Management. *Assembly Engineering*, Sept, pp.20-25.

Mansoor, E. M., 1964, Assembly Line Balancing-Improvement of Ranked Positional Weight Technique. *Journal of Industrial Engineering*, v. 15 n. 2, Mar-Apr, pp.73-77.

Moodie, C. L., Young, H. H., 1965, Heuristic Method of Assembly Line Balancing for

Assumptions of Constant or Variable Work Element Times. *Journal of Industrial Engineering*, v. 16 n. 1, Jan-Feb, pp.23-29.

Okamura, K., and **Yamashina, H.**, 1979, A Heuristic Algorithm for the Assembly line Model-Mix sequencing Problem to minimize the risk of stopping the Conveyor. *International Journal of Production Research*, v. 17 n. 3, pp.233-247.

Salveson, M. E., 1955, Assembly Line Balancing Problem. *Journal of Industrial Engineering*, May-June, pp.18-25.

Sarin, S. C., and **Erel, E.**, 1989, A Methodology for solving Single-Model, Stochastic Assembly Line Balancing Problem. *Working Paper*

Sarin, S. C., and **Erel, E.**, 1990, Development of Cost Model for the Single-Model Stochastic Assembly Line Balancing Problem. *International Journal of Production Research*, v. 28 n. 7, pp.1305-1316.

Smith, P. R., 1990, A Computerized Search Methodology for the Design of Mixed Model Assembly Systems. *Master's Thesis*, Virginia Polytechnic Institute and State University, 1990.

Sphicas, G. P., **Silverman, F. N.**, 1976, Deterministic equivalents for stochastic assembly line balancing. *AIIE Transactions*, v. 8, n. 2, June, pp.280-282.

Thomopoulos, N. T., 1967, Line Balancing-Sequencing for Mixed-Model Assembly. *Management Science*, v. 14 n. 2, Oct, pp.B-59-75.

Thomopoulos, N. T., 1970, Mixed Model Line Balancing with smoothed Station Assignments. *Management Science*, v. 16 n. 9, May, pp.593-603.

Tonge, F. M., 1960, Summary of a Heuristic Line Balancing Procedure. *Management Science*,

v. 7, n. 1, pp.21-39.

Tonge, F. M., 1961, A heuristic program for assembly line balancing. *Prentice-Hall*, Inc., New Jersey.

Vrat, P., Virani, A., 1976, 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*, v. 14, n. 4, pp.445-463.

Wester, L., Kilbridge, M., 1964, The Assembly Line Model-Mix Sequencing Problem. *Proceedings of the Third International Conference of Operations Research*, pp.247-254.

APPENDIX A: COMPUTER PROGRAM LISTINGS

*** SOURCE CODE FOR GENERATION OF BALANCES ***

```
#include <stdio.h>
#include <math.h>
#include <conio.h>
#include <time.h>

        /** UPPER LIMITS FOR ARRAY DIMENSIONS **/
#define MAXELS    75          /* elements (60) */
#define MAXMODS   12          /* models  (12) */
#define MAXSTAT   36          /* stations (30) */
#define MAXSOLS   100         /* solutions    */
#define MAXSCHEDS 850         /* schedules    */
#define MAXPRODS  12                  /* products (12) */

              /**** GLOBAL VARIABLES ****/
 /** FILENAMES OF INPUT, RUN AND OUTPUT FILE INPUT FROM COMMAND LINE
   **/
char *filein, *filerun, *fileout ;

        /** PROBLEM RELATED GLOBAL VAR'S TO BE INPUT FROM INPUT FILE **/
int elements , models ;               /* number of elements and models */
int prec [MAXELS][MAXELS] ; /* precedence between elements */
float wk_el_dis [MAXELS][MAXMODS] ;   /* pr. dist. type per element, model */
float wk_el_avg [MAXELS][MAXMODS] ;   /* average elemental times per model */
float wk_el_std [MAXELS][MAXMODS] ;   /* Std. dev. for el. times per model */
int batch_size [MAXMODS] ;      /* batch sizes per model */
float labor_cost ;            /* uniform labor cost */
float inc_cost [MAXELS] ;           /* incompletion costs per element */

        /** RUN RELATED GLOBAL VAR'S TO BE INPUT FROM RUN FILE **/
int horizon ;                 /* # of stations to look ahead */
long int break_seq_2 ;          /* max solutions generated for horizon */
long int break_seq_1 ;          /* max solutions generated per solution */
float utol , ltol ;           /* acceptable percentages for fluctuation */
                              /* above, below ideal station time */
float mtime ;                 /* max. cycle time (if 0 --> ideal) */
int pr ;                  /* 1 to print all sequences 0 to omit */
int stations , solutions ;    /* max number of stations and solutions */
float opal ;                  /* operator allowance */

        /** GLOBAL VAR'S TO BE MAINTAINED BY PROGRAM **/
long int total_seq_1 , total_seq_2 , last_sol ;          /* # of generated balances */
int lstat ;      /* last station to balance */
float twc , m_delta [MAXMODS] ; /* total work content, ideal service time
                                                        for each model
                                                        per station */
float ttcll , ttcul ;         /* lower , upper limit for station time */
```

```
FILE *frun , *fin , *fout ; /* run, input and output file */
float eltime [MAXELS];       /* total element time for all models */
float rpw [MAXELS] ;         /* ranked positional weight */
int max_st [MAXELS] ;     /* max allowable station for el's based on rpw */
int best_list [MAXELS+1][MAXSOLS] ;  /* list with best solutions */
int best_listg [MAXSTAT+2][MAXSOLS] ;  /* vector with best solutions */
float best_d[MAXSOLS],big_d ;  /* value of delta till current iteration */
int low_best = 0 ;        /* position of lowest solution among best */

                  /**** PROGRAM FUNCTIONS ****/
void input_data ()       /** INPUT BALANCING DATA FROM INPUT FILES **/
{
 int i , j , k , l ;
                  /* Input data from input file */
 fin = fopen (filein , "r") ;
 fscanf (fin , "%d " , &elements) ;
 fscanf (fin , "%d " , &models) ;
 for (i=0 ; i < elements ; i++)
 { fscanf (fin , "%d " , &k) ;
  for (j=0 ; j<elements ; j++) prec [j][i] = 0 ;
  for (l=0 ; l<k ; l++)
  { fscanf (fin , "%d " , &j) ;
   prec [j-1][i] = 1 ;
  }
 }
 for (i=0 ; i < elements ; i++)
 { for (j=0 ; j < models ; j++)
  { fscanf (fin , "%f " , &wk_el_dis [i][j]) ;
   fscanf (fin , "%f " , &wk_el_avg [i][j]) ;
   fscanf (fin , "%f " , &wk_el_std [i][j]) ;
  }
 }
 for (j=0 ; j < models ; j++)
  fscanf (fin , "%d " , &batch_size [j]) ;
 fclose (fin) ;
                  /* Input data from run file */
 frun = fopen (filerun , "r") ;
 fscanf (frun , "%d %ld %ld " , &horizon , &break_seq_2 , &break_seq_1) ;
 fscanf (frun , "%f %f %f " , &utol , &ltol , &mtime) ;
 fscanf (frun , "%d %d %d ",&pr , &stations , &solutions) ;
 fscanf (frun , "%f " , &opal) ;
 fclose (frun) ;
}

void prep_data ()            /** PREPARE DATA **/
                             /* compute values for ideal station time,
                                    its upper and lower limits, m_delta
                                    and eltime */
{
 int i , j ;
 int k , last ;
```

```
                                /* make precedence matrix denser for
                                   faster results */
  big_d = 0;
  for (last=elements; last>elements-1; last--)
  { for (i=0; i<last-2; i++)
    { for (j=i+1; j<last-1; j++)
      { if (prec [i][j])
          { for (k=j+1; k<last; k++)
              if (prec [j][k]) prec [i][k] = 1;
          }
      }
    }
  }
  twc = 0;                      /** INITIAL VALUES **/
  for (i=0;i < elements; i++)
  { eltime [i] = 0;
    for (j=0; j < models; j++)
    { eltime [i] += batch_size [j] * wk_el_avg [i][j];
      m_delta [j] += wk_el_avg [i][j];
    }
    twc += eltime [i];
  }
  for (i=0;i < elements; i++)
  { rpw [i] = eltime [i];
    for (j=i+1; j<elements; j++)
      if (prec [i][j]) rpw [i] += eltime [j];
  }
  for (j=0; j < models; j++)
  { m_delta [j] = m_delta [j] * batch_size [j];
    big_d += m_delta [j];
  }
  init_best ();
  if (mtime == 0) mtime = twc / stations;
  if (ltol == 0) ltol = 3 - 3 * twc / (stations * mtime);
  if (horizon == 0) horizon = stations;
  ttcll = mtime * (1 - ltol);
  ttcul = mtime * (1 + utol);
  if (opal == 0) opal = ttcul;
  fprintf (fout,"\n Cycle Time              %4f\n", mtime);
  fprintf (fout,"\n Lower limit for station time %4f", ttcll);
  fprintf (fout,"\n Upper limit for station time %4f\n", ttcul);
  fprintf (fout,"\n Maximum Operator Allowance  %4f\n\n", opal);
  for (i=0; i < elements; i++)
    max_st [i] = stations;
}

                                                  /** FIND BALANCE **/
void find_balance (list, stime, listg)
                        /* find all possible sequences starting with list list;
                             make all following sequences inherit appropriate
                             values of station time and lists list and listg */
```

```
int list [] , listg [] ;
float stime ;
{
 int elem , j , first , first2 , check_dup ;
 int listg1 [MAXSTAT+2] , listg2 [MAXSTAT+2] ;
 int sub_list [MAXELS+1] , sub_list2 [MAXELS+1] ;
 float stime1 , stime2;
 first = list [list [0]] ;    /* initial values */
 stime2 = 0 ;
 check_dup = 0 ;
 if (total_seq_2 >= break_seq_2 || total_seq_1 >= break_seq_1) return ;
 if ((stime >= ttcll && stime <= ttcul) || list [0] == elements)
 {
  stime2 = stime ;          /* copy values for additional elements to */
                                    /* the same station  */
  copy_list (list , sub_list2) ;
  copy_vector (listg , listg2 , stations + 2) ;
  first2 = first ;
  stime = 0 ;               /* prepare for new station */
  listg [++listg [0]] = list [0] ;
  if (listg [0] > lstat) return ;
  listg [listg [0] + 1] = 0 ;
  first = 0 ;
  if (duplicate (list,listg)) check_dup=1 ;
  /* check for complete balance */
  if (!check_dup && (listg [0] == lstat && lstat != stations) ||
    (list [0] == elements))
  { eval_list (list , listg) ;
    return ;
  }
 }
 if (!check_dup)            /* add additional elements to balance */
 {
 for (elem = first ; elem < elements ; elem++)  /* candidate elements */
    add_el (elem , list , listg , stime) ;
 }
 if (stime2 != 0)
 { first = first2 ;
  for (elem = first ; elem < elements ; elem++)
    add_el (elem , sub_list2 , listg2 , stime2) ;
 }
}

                                        /** ADD AN ELEMENT TO BALANCE **/
add_el (elem , list , listg , stime)
                /* Create a new copy of relevant variables
                i.e. list, listg , stime.
                If it is legal, add to the copy element elem.
                Balance again the new set of variables */
int list [] ;
int elem ;
```

```
int listg [] ;
float stime ;
{
 int sub_list [MAXELS+1] , listg1 [MAXSTAT+2] , l , m , k ;
 float stime1 , mod_time , mod_t = 0 ;
 copy_list (list , sub_list) ;
 copy_vector (listg , listg1 , stations + 2) ;
 stime1 = stime ;
 k = listg [0] ;
 for (m = 0 ; m < models ; m++)   /* Calculate assigned time per model */
 { mod_time = 0 ;
  for (l = first_el (k , listg) ; l <= last_el (k , listg) ; l++)
   mod_time += wk_el_avg [l][m] ;
  if (mod_time > mod_t) mod_t = mod_time ;
 }
                                        /* Check necessary conditions */
 if (!in_list (sub_list , elem) &&    /* element not previously assigned */
  !el_precedes (elem , sub_list) &&  /* precedence constraints */
  stime1 + eltime [elem] <= ttcul &&  /* enough time in station */
  mod_t <= opal &&                    /* enough operator allowance */
  max_st [elem] >= listg [0])      /* enough space for followers*/
 { stime1 += eltime [elem] ;
  sub_list [++sub_list [0]] = elem ;
  find_balance (sub_list , stime1 , listg1) ;
 }
 return ;
}

                                  /** EVALUATE FEASIBLE BALANCE FOUND **/
eval_list (list , listg)  /* compare current delta value to optimal
                             so far; if current is better update
                             best_list, best_listg and best_d;
                             if pr is 1 print intermediate balance;
                             calculate delta values and station times */
int list [] , listg [] ;
{
 int i , j , stat , start1 ;
 float stime1 = 0 , delt1 [MAXMODS] , deltt , deltot , ttot ;
 total_seq_1++ ;
 total_seq_2++ ;
 gotoxy (1,3) ;
 printf("Sequence for current solution  %ld  ",total_seq_1);
 if (pr) fprintf (fout,"\nSeq. %ld :\n" , total_seq_2) ;
 start1 = 0 ;
 deltot = 0 ;
 ttot = 0 ;
 for (stat = 0 ; listg [stat + 1] != 0 ; stat ++)
 { deltt = 0 ;
  stime1 = 0 ;
  for (j=0 ; j<models ; j++)
   delt1 [j] = 0 ;
```

```
   if (pr) fprintf (fout,"Station %d : " , stat + 1) ;
   for (i = start1 ; i < listg [stat + 1] ; i ++)
   { stime1 += eltime [list [i+1]] ;
     if (pr) fprintf (fout,"%4d" , list [i+1] + 1) ;
     for (j=0 ; j<models ; j++)
          delt1 [j] += wk_el_avg [list [i+1]] [j] ;
   }
   start1 = listg [stat + 1] ;
   for (j=0 ; j<models ; j++)
     deltt += fabs (m_delta [j]/listg[0] - batch_size [j]*delt1 [j]);
   if (pr) fprintf (fout,"    Time : %8.2f  \(D = %6.2f\)\n",stime1,deltt) ;
   if (deltt > deltot) deltot = deltt ;    /* max delta instead of sum */
   ttot += fabs (stime1 - mtime) ;
  }
  if (pr) fprintf (fout,"    D = %7.2f\n" , deltot) ;
  if (ttot < best_d [low_best])
  { best_d [low_best] = deltot ;
    copy_best (list , listg) ;
    find_new_low () ;
  }
  return ;
}

sort_solutions ()
{
  int i , j , k , sc ;
  float dv ;
  gotoxy (1 , 8) ;
  puts ("Sorting Solutions...") ;
  for (i = 0 ; i < last_sol-1 ; i++)
    for (j = i+1 ; j < last_sol ; j++)
    { if (best_d [j] < best_d [i])
      { dv = best_d [i] ;
          best_d [i] = best_d [j] ;
          best_d [j] = dv ;
          for (k = 1 ; k <= best_list [0][i] ; k++)
            { sc = best_list [k][i] ;
              best_list [k][i] = best_list [k][j] ;
              best_list [k][j] = sc ;
            }
          for (k = 1 ; k <= best_list [0][i] ; k++)
            { sc = best_listg [k][i] ;
              best_listg [k][i] = best_listg [k][j] ;
              best_listg [k][j] = sc ;
            }
      }
    }
}

                                        /** PRINT THE BEST BALANCE **/
print_best ()  /* print the best overall balance in list
```

```
                                and listg; calculate station times and
                                delta values */
{
 int i , j , k , stat , start1 ;
 float stime1 , delt1 [MAXMODS] , deltt , tt, deltot , ttot ;
 last_sol = imin (solutions , total_seq_2) ;
 puts ("Saving Report on Generated Balances...") ;
 fprintf (fout , "\n\nThe %d best balances " , last_sol) ;
 fprintf (fout , "out of %ld investigated " , total_seq_2) ;
 fprintf(fout,"\nin terms of lowest Delta are : \n\n") ;
 for (k = 0 ; k < last_sol ; k++)
 { stime1 = 0 ;
  start1 = 0 ;
  deltot = 0 ;
  ttot = 0 ;
  fprintf (fout,"Solution %d\n" , k + 1) ;
  for (stat = 0 ; best_listg [stat + 1][k] != 0 ; stat ++)
  { deltt = 0 ;
   stime1 = 0 ;
   for (j=0 ; j<models ; j++) delt1 [j] = 0 ;
   fprintf (fout,"Station %d :" , stat + 1) ;
        for (i = start1 ; i < best_listg [stat + 1][k] ; i ++)
   { stime1 += eltime [best_list [i+1][k]] ;
        fprintf (fout,"%3d" , best_list [i+1][k] + 1) ;
        for (j=0 ; j<models ; j++)
         delt1 [j] += wk_el_avg [best_list [i+1][k]] [j] ;
         tt = fabs (stime1 - mtime) ;
   }
   fprintf (fout,"  Time :%8.2f Dif :%8.2f " , stime1 , tt) ;
   start1 = best_listg [stat + 1][k] ;
   for (j=0 ; j<models ; j++)
        deltt += fabs (m_delta [j]/best_listg[0][k] - batch_size [j]*delt1 [j]);
   fprintf (fout," \(D = %6.2f\)\n" , deltt) ;
   if (deltt > deltot) deltot = deltt ;
   ttot += tt ;
  }
  fprintf (fout,"With a maximum Delta value of %7.2f" , deltot) ;
  fprintf (fout," and a total difference of %7.2f\n\n" , ttot) ;
 }
 return ;
}

retrieve (list , listg , i)
int list [] , listg [] , i ;
{
 int j , k ;
 FILE *ftemp ;
 ftemp = fopen ("bal.tmp" , "r") ;
 for (j = 0 ; j <= i ; j++)
 { fscanf (ftemp , "%d " , &list [0]) ;
  for (k = 1 ; k <= list [0] ; k++)
```

```
    { fscanf (ftemp , "%d " , &list [k]) ;
    }
    for (k = 0 ; k < stations+2 ; k++)
      fscanf (ftemp , "%d " , &listg [k]) ;
  }
  listg[0]-- ;
  fclose (ftemp) ;
  return ;
}

save_solutions ()
{
  int i , j ;
  FILE *ftemp ;
  puts ("Saving Balances...") ;
  ftemp = fopen ("bal.tmp" , "w") ;
  for (i = 0 ; i < solutions ; i++)
  { fprintf (ftemp , "%d " , best_list [0][i]) ;
    for (j = 1 ; j <= best_list [0][i] ; j++)
      fprintf (ftemp , "%d " , best_list [j][i]) ;
    for (j = 0 ; j < stations+2 ; j++)
      fprintf (ftemp , "%d " , best_listg [j][i]) ;
    fputs ("\n" , ftemp) ;
  }
  fclose (ftemp) ;
  return ;
}

int duplicate (list , listg)  /* true if current balance duplicates */
int list [] , listg [] ;      /*  another balance supposed to have */
{                             /*  been generated sooner */
  int st , last_st ;
  last_st = listg [0] ;
  for (st = last_st - 1 ; st >= 1 ; st--)
    if (list [first_el (st , listg)] > list [first_el (last_st , listg)] &&
    !st_precedes (st , last_st , list , listg)) return 1 ;
  return 0 ;
}

int st_precedes (k , l , list , listg)  /* true if there is any element */
int k , l , list [] , listg [] ;        /*  in station k that precedes */
{                                       /*  any element in station l */
  int i , j ;
  for (i = first_el (k , listg) ; i <= last_el (k , listg) ; i++)
  { for (j = imax (first_el (l,listg) , i+1) ; j <= last_el (l,listg) ; j++)
      if (prec [list [i]][list [j]]) return 1 ;
  }
  return 0 ;
}

int first_el (k , listg)  /* first (lowest) element assigned to station k */
```

```
int k , listg [] ;
{
 if (k = = 1) return 1 ;
 return listg [k-1] + 1 ;
}

int last_el (k , listg)    /* last (highest) element assigned to station k */
int k , listg [] ;
{
 return listg [k] ;
}

float stat_time (s , list , listg)
int s , list [] , listg [] ;
{
 float stime = 0 ;
 int k ;
 for (k=first_el(s,listg) ; k< =last_el(s,listg) ; k++)
 stime += eltime [list [k]] ;
 return stime ;
}

copy_best (list , listg)   /*  */
int list [] , listg [] ;
{
 int i ;
 for (i=0 ; i<list [0] + 1 ; i++) best_list [i][low_best] = list [i] ;
 for (i=0 ; i<stations+2 ; i++)
   best_listg [i][low_best] = listg [i] ;
 return ;
}

copy_from_best (list , listg)   /*  */
int list [] , listg [] ;
{
 int i ;
 for (i=0 ; i<list [0] + 1 ; i++) list [i] = best_list [i][low_best] ;
 for (i=0 ; i<stations+2 ; i++)
   listg [i] = best_listg [i][low_best] ;
 return ;
}

init_best ()
{
 int i , j ;
 total_seq_2 = 0 ;
 for (i = 0 ; i < solutions ; i++)
 { best_d [i] = 2 * stations * big_d ;
   best_list [0][i] = 0 ;
   for (j = 0 ; j < stations+2 ; j++)
     best_listg [j][i] = 0 ;
```

```
 }
 low_best = 0;
 return;
}

copy_vector (a1, a2, n)  /* put a copy of vector a1 to a2 of size n */
int a1 [], a2 [], n;
{
 int i;
 for (i=0; i<n; i++)
   a2 [i] = a1 [i];
}

find_new_low ()
{
 int j;
 low_best = 0;
 for (j = 1; j < solutions; j++)
   if (best_d [j] > best_d [low_best]) low_best = j;
}

int el_precedes (j,list)     /* true if element j has precedents */
int j, list [];              /* that are not part of the list */
{
 int i;
 for (i=0; i<j; i++)
   if ( (!in_list (list, i)) && prec [i] [j] ) return 1;
 return 0;
}



copy_list (a1, a2)       /* put a copy of list a1 to list a2 */
int a1 [], a2 [];
{
 int i;
 for (i=0; i<a1 [0] + 1; i++) a2 [i] = a1 [i];
 return;
}

int in_list (list, k)        /* true if element k is part of list */
int list [], k;
{
 int i;
 for (i=0; i < list [0]; i++)
 if (list [i+1] == k) return 1;     /*  return i + 1 ??? */
 return 0;
}

int imax (i, j)              /* maximum of integers i, j */
int i, j;
```

```
{
 if (i > j) return i ;
 return j ;
}

int imin (i , j)              /* maximum of integers i , j */
int i , j ;
{
 if (i < j) return i ;
 return j ;
}

int sum (vector , n)                    /* sum of n first elements of vector */
int vector [] , n ;
{
int i , s ;
for (i = 0 , s = 0 ; i < n ; s += vector [i] , i++) ;
return s;
}

void prtime ()              /* print the current date and time */
{
 time_t tnow ;
 time (&tnow) ;
 fprintf (fout,ctime (&tnow)) ;
}

                                        /** MAIN PROGRAM **/
main (int argc , char *argv [])
{
 int list [MAXELS + 1] , listg [MAXSTAT + 2] , i , k ;
 float stime ;
 clrscr () ;
 if (argc != 4)
 { printf ("\nThe balancing program requires 3 arguments! \n") ;
   return ;
 }
 filein = argv [1] ;
 filerun = argv [2] ;
 fileout = argv [3] ;
 for (i = 0 ; i < elements + 1 ; list [i] = 0 , i++) ;
 for (i = 0 ; i < stations + 2 ; listg [i] = 0 , i++) ;
 input_data () ;
 fout = fopen (fileout , "w") ;
 fprintf (fout,"Starting time ") ;
 prtime () ;
 prep_data () ;
 printf ("Generating Balances...") ;
 lstat = horizon ;
 gotoxy (1 , 4) ;
 printf ("  Max Station = %d    " , lstat) ;
```

```
  find_balance (list , 0 , listg) ;
  last_sol = imin (solutions , total_seq_2) ;
  sort_solutions () ;
  save_solutions () ;
  while (lstat < stations && total_seq_2 != 0)
  { gotoxy (1 , 4) ;
    printf ("Previous solutions = %ld   " , total_seq_2) ;
    init_best () ;
    lstat += horizon ;
    if (lstat > stations) lstat = stations ;
    printf ("   Max Station = %d     " , lstat) ;
    for (i = 0 ; i < solutions && total_seq_2 < break_seq_2 ; i++)
    { retrieve (list , listg , i) ;
      gotoxy (1 , 5) ;
      printf ("Saved Solution = %d     " , i + 1) ;
      total_seq_1 = 0 ;
      gotoxy (1 , 6) ;
      printf("Overall Sequences %ld   ",total_seq_2);
      stime = stat_time (listg[0]+1 , list , listg) ;
      if (list [0] != 0) find_balance (list , stime , listg) ;
    }
    last_sol = imin (solutions , total_seq_2) ;
    sort_solutions () ;
    save_solutions () ;
  }
  print_best () ;
  fprintf (fout,"End Time ") ;
  prtime () ;
  fclose (fout) ;
}
```

*** SOURCE CODE FOR GENERATION OF SCHEDULES ***

```
#include <stdio.h>
#include <math.h>
#include <conio.h>
#include <time.h>

        /** UPPER LIMITS FOR ARRAY DIMENSIONS **/
#define MAXELS    75          /* elements  (75) */
#define MAXMODS   12          /* models    (12) */
#define MAXSCHEDS 850         /* schedules (850) */
#define MAXPRODS  12                  /* products  (12) */

                /**** GLOBAL VARIABLES ****/

 /** FILENAME OF INPUT FILE TO BE INPUT FROM COMMAND LINE **/
char *filein ;

        /** PROBLEM RELATED GLOBAL VAR'S TO BE INPUT FROM INPUT FILE **/
int elements , models ;             /* number of elements and models */
float wk_el_avg [MAXELS][MAXMODS] ;  /* average elemental times per model */
int batch_size [MAXMODS] ;      /* batch sizes per model */

        /** RUN RELATED GLOBAL VAR'S TO BE INPUT FROM RUN FILE **/
float mtime ;                       /* avg. cycle time (if 0 --> ideal) */

        /** GLOBAL VAR'S TO BE MAINTAINED BY PROGRAM **/
float twc ;                         /* total work content */
FILE *fin ;                         /* input file */
float eltime [MAXMODS] ;        /* total element time per model */
int scheds ;                        /* no. of model schedules */
int sc_prec [MAXPRODS][MAXPRODS] ; /* artificial precedences between models */
int sched [MAXPRODS][MAXSCHEDS] ;          /* list with schedules */
int product [MAXPRODS] ;        /* unscheduled models */
int bcd ;                 /* bcd of all batch sizes */
int small_model ;                   /* model with least demand */
int products ;                      /* total scheduled quantity of all models */
float dev [MAXSCHEDS] ;                   /* deviation from average model time */

                /**** PROGRAM FUNCTIONS ****/

void input_data ()      /** INPUT BALANCING DATA FROM INPUT FILES **/
{
 int i , j , k , l ;
                /* Input data from input file */
 fin = fopen (filein , "r") ;
 fscanf (fin , "%d " , &elements) ;
```

```
 fscanf (fin , "%d " , &models) ;
 for (i=0 ; i < elements ; i++)
 { fscanf (fin , "%d " , &k) ;
  for (l=0 ; l<k ; l++)
   fscanf (fin , "%d " , &j) ;
 }
 for (i=0 ; i < elements ; i++)
  for (j=0 ; j < models ; j++)
   fscanf (fin , "%*f %f %*f " , &wk_el_avg [i][j]) ;
 for (j=0 ; j < models ; j++)
  fscanf (fin , "%d " , &batch_size [j]) ;
 fclose (fin) ;
                  /* Input data from run file */
}

void prep_data ()              /** PREPARE DATA **/
                                  /* compute values for ideal station time,
                                   its upper and lower limits, model_delta
                                   and eltime */
{
 int i , j , factor ;
 int k , last , found ;
 twc = 0 ;                      /** INITIAL VALUES **/
 for (i=0 ;i < models ; i++)
 { eltime [i] = 0 ;
  for (j=0 ; j < elements ; j++)
   eltime [i] += wk_el_avg [j][i] ;
  twc += eltime [i] ;
 }
 mtime = twc / models ;
 small_model = 0 ;                          /* find model with less demand */
 for (i = 1 ; i < models ; i ++)
  if (batch_size [i] < batch_size [small_model]) small_model = i ;
 bcd = 0 ;                  /* find biggest common divisor of sched. quantities */
 for (k = batch_size [small_model] ; bcd == 0 ; k--)
 { found = 1 ;
  for (i = 0 ; i < models ; i++)
   if (fmod(batch_size [i],k)) found = 0 ;
  if (found) bcd = k ;
 }
 last = 0 ;                                 /* prepare data for scheduling */
 products = sum (batch_size , models) ;
 for (i=0 ; i < products/bcd ; i++)
  for (j= 0 ; j < products/bcd ; j++) sc_prec [i][j] = 0 ;
 for (i=0 ; i < models ; i++)
 { product [last] = i ;
  for (j=last+1 ; j < last + batch_size [i] / bcd ; j++)
  { sc_prec [j-1][j] = 1 ;
   product [j] = i ;
  }
  last += batch_size [i] / bcd ;
```

```
 }
}

void find_schedules ()
                    /* find all possible sequences starting with list list;
                         make all following sequences inherit appropriate
                         values of station time and lists list and listg */
{
 int list [MAXPRODS] ;
 int mod , i , j ;
 puts ("Generating Schedules...") ;
 list [0] = 1 ;      /* initial values */
 list [1] = sum (batch_size , small_model) / bcd ;
 for (i = 2 ; i < = products / bcd ; list [i] = 0 , i++) ;
 scheds = 0 ;
 find_sched (list) ;
 printf ("%d Schedules were generated.\n" , scheds) ;
 return ;
}

find_sched (list)
int list [] ;
{
 int i ;
 float st ;
 if (list [0] = = products / bcd)
 { scheds++ ;
   dev [scheds-1] = 0 ; st = 0 ;
   for (i = 0 ; i < = list [0] ; i++)
   { sched [i][scheds-1] = list [i] ;
     st += eltime [product [list[i]]] ;
     dev [scheds-1] += fabs ((i+1) * mtime - st) ;
   }
   return ;
 }
 for (i = 0 ; i < products / bcd ; i++)   /* candidate products */
   add_mod (i , list) ;
 return ;
}

                                        /** ADD A MODEL TO SEQUENCE **/
add_mod (mod , list)
int list [] ;
int mod ;
{
 int sub_list [MAXPRODS + 1] ;
 copy_list (list , sub_list) ;
 if (!in_list (sub_list , mod) && !pr_precedes (mod , sub_list))
 { sub_list [++sub_list [0]] = mod ;
   find_sched (sub_list) ;
 }
```

```
}

int pr_precedes (j , list)
int j , list [] ;
{
 int i ;
 for (i = 0 ; i < j ; i ++)
  if ((!in_list (list , i)) && sc_prec [i][j]) return 1 ;
 return 0 ;
}

save_schedules ()
{
 int i , j ;
 FILE *ftemp ;
 puts ("Saving Schedules...") ;
 ftemp = fopen ("seq.tmp" , "w") ;
 for (i = 0 ; i < scheds ; i++)
 { fprintf (ftemp , "%d " , sched [0][i]) ;
  for (j = 1 ; j <= sched [0][i] ; j++)
   fprintf (ftemp , "%d " , sched [j][i]) ;
  fputs ("\n" , ftemp) ;
 }
 fclose (ftemp) ;
 return ;
}

sort_schedules ()
{
 int i , j , k , sc ;
 float dv ;
 puts ("Sorting Schedules...") ;
 for (i = 0 ; i < scheds-1 ; i++)
  for (j = i+1 ; j < scheds ; j++)
  { if (dev [j] < dev [i])
   { dv = dev [i] ;
        dev [i] = dev [j] ;
        dev [j] = dv ;
        for (k = 1 ; k <= sched [0][i] ; k++)
         { sc = sched [k][i] ;
          sched [k][i] = sched [k][j] ;
          sched [k][j] = sc ;
         }
   }
  }
}

copy_list (a1 , a2)      /* put a copy of list a1 to list a2 */
int a1 [] , a2 [] ;
{
 int i ;
```

```
 for (i=0 ; i<a1 [0] + 1 ; i++) a2 [i] = a1 [i] ;
 return ;
}

int in_list (list , k)          /* true if element k is part of list */
int list [] , k ;
{
 int i ;
 for (i=0 ; i < list [0] ; i++)
 if (list [i+1] = = k) return 1 ;
 return 0 ;
}

int sum (vector , n)                    /* sum of n first elements of vector */
int vector [] , n ;
{
int i , s ;
for (i = 0 , s = 0 ; i < n ; s + = vector [i] , i++) ;
return s;
}

                                        /** MAIN PROGRAM **/
main (int argc , char *argv [])
{
 clrscr () ;
 if (argc != 4)
 { printf ("The sequencing program requires 3 arguments! \n") ;
   return ;
 }
 filein = argv [1] ;
 puts ("Reading and Preparing Data...") ;
 input_data () ;
 prep_data () ;
 find_schedules () ;
 sort_schedules () ;
 save_schedules () ;
}
```

*** SOURCE CODE FOR COST EVALUATION ***

```
#include <stdio.h>
#include <math.h>
#include <conio.h>
#include <time.h>

        /** UPPER LIMITS FOR ARRAY DIMENSIONS **/
#define MAXELS    75            /* elements  (75) */
#define MAXMODS   12            /* models    (12) */
#define MAXSTAT   30            /* stations  (30) */
#define MAXSOLS   50            /* solutions (10) */
#define MAXSCHEDS 850           /* schedules (850) */
#define MAXPRODS  12                    /* products  (12) */

                /**** GLOBAL VARIABLES ****/

 /** FILENAMES OF INPUT, RUN AND OUTPUT FILE INPUT FROM COMMAND LINE
   **/
char *filein, *filerun, *fileout ;

        /** PROBLEM RELATED GLOBAL VAR'S TO BE INPUT FROM INPUT FILE **/
int elements , models ;                 /* number of elements and models */
int prec [MAXELS][MAXELS] ; /* precedence between elements */
float wk_el_dis [MAXELS][MAXMODS] ;   /* pr. dist. type per element, model */
float wk_el_avg [MAXELS][MAXMODS] ;   /* average elemental times per model */
float wk_el_std [MAXELS][MAXMODS] ;   /* Std. dev. for el. times per model */
int batch_size [MAXMODS] ;      /* batch sizes per model */
float labor_cost ;              /* uniform labor cost */
float inc_cost [MAXELS] ;               /* incompletion costs per element */

        /** RUN RELATED GLOBAL VAR'S TO BE INPUT FROM RUN FILE **/
float utol , ltol ;             /* acceptable percentages for fluctuation */
                                        /* above, below ideal station time */
float mtime ;                           /* max. cycle time (if 0 --> ideal) */
int pr ;                    /* 1 to print all sequences 0 to omit */
int stations , solutions ;    /* max number of stations and solutions */
float opal ;                            /* operator allowance */

        /** GLOBAL VAR'S TO BE MAINTAINED BY PROGRAM **/
long last_sol ;                 /* # of generated balances */
float twc , model_delta [MAXMODS] ; /* total work content, ideal service time
                                                                for each model
                                                                per station */
float ttcll , ttcul ;           /* lower , upper limit for station time */
FILE *frun , *fin , *fout ; /* run, input and output file */
float eltime [MAXELS];          /* total element time for all models */
```

```
float rpw [MAXELS] ;        /* ranked positional weight */
int max_st [MAXELS] ;    /* max allowable station for el's based on rpw */
int best_list [MAXELS+1][MAXSOLS] ;  /* list with best solutions */
int best_listg [MAXSTAT+2][MAXSOLS] ;  /* vector with best solutions */
float best_c[MAXSOLS],big_c ;  /* value of delta till current iteration */
int best_sched [MAXPRODS][MAXSOLS] ;
int low_best = 0 ;        /* position of lowest solution among best */
int scheds ;                        /* no. of model schedules */
int sched [MAXPRODS][MAXSCHEDS] ;          /* list with schedules */
int product [MAXPRODS] ;        /* unscheduled models */
float d [MAXSTAT+1][MAXPRODS+1] ;
int bcd ;                /* bcd of all batch sizes */
int products ;                      /* total scheduled quantity of all models */
int list [MAXELS+1] ;
int listg [MAXSTAT+2] ;
int mlist [MAXELS+1][MAXMODS] ;
int listh [MAXELS+2] ;
float normtabl [350] ;
int cur_mod , cur_sc , best_found , part ;
int saved_bal , eval_solutions ;

                /**** PROGRAM FUNCTIONS ****/

void input_data ()       /** INPUT BALANCING DATA FROM INPUT FILES **/
{
 int i , j , k , l ;
                /* Input data from input file */
 fin = fopen (filein , "r") ;
 fscanf (fin , "%d " , &elements) ;
 fscanf (fin , "%d " , &models) ;
 for (i=0 ; i < elements ; i++)
 { fscanf (fin , "%d " , &k) ;
  for (j=0 ; j<elements ; j++) prec [j][i] = 0 ;
  for (l=0 ; l<k ; l++)
  { fscanf (fin , "%d " , &j) ;
   prec [j-1][i] = 1 ;
  }
 }
 for (i=0 ; i < elements ; i++)
 { for (j=0 ; j < models ; j++)
  { fscanf (fin , "%f " , &wk_el_dis [i][j]) ;
   fscanf (fin , "%f " , &wk_el_avg [i][j]) ;
   fscanf (fin , "%f " , &wk_el_std [i][j]) ;
  }
 }
 for (j=0 ; j < models ; j++)
  fscanf (fin , "%d " , &batch_size [j]) ;
 fscanf (fin , "%f " , &labor_cost) ;
 for (i=0 ; i < elements ; i++)
  fscanf (fin , "%f " , &inc_cost [i]) ;
 fclose (fin) ;
```

```
                /* Input data from run file */
  frun = fopen (filerun , "r") ;
  fscanf (frun , "%*d %l*d %l*d ") ;
  fscanf (frun , "%f %f %f " , &utol , &ltol , &mtime) ;
  fscanf (frun , "%d %d %d " , &pr , &stations , &solutions) ;
  fscanf (frun , "%f " , &opal) ;
  fclose (frun) ;
  frun = fopen ("normal.dat" , "r") ;
  for (i=0 ; i < 350 ; i++) fscanf (frun , "%f " , &normtabl [i]) ;
  fclose (frun) ;
  return ;
}

void prep_data ()                    /** PREPARE DATA **/
                                     /* compute values for ideal station time,
                                      its upper and lower limits, model_delta
                                      and eltime */
{
  int i , j , factor ;
  int k , last , found ;
  big_c = 5 * stations * labor_cost ;

                                    /* make precedence matrix denser for
                                      faster results */
  for (last=elements ; last>elements-1 ; last--)
  { for (i=0 ; i<last-2 ; i++)
    { for (j=i+1 ; j<last-1 ; j++)
      { if (prec [i][j])
          { for (k=j+1 ; k<last ; k++)
              if (prec [j][k]) prec [i][k] = 1 ;
          }
      }
    }
  }
  twc = 0 ;                          /** INITIAL VALUES **/
  for (i=0 ;i < elements ; i++)
  { eltime [i] = 0 ;
    for (j=0 ; j < models ; j++)
    { eltime [i] += batch_size [j] * wk_el_avg [i][j] ;
      model_delta [j] += wk_el_avg [i][j] ;
    }
    twc += eltime [i] ;
  }
  init_best () ;
  products = sum (batch_size , models) ;
  bcd = products / sched [0][0] ;
  if (mtime == 0) mtime = twc / (stations * products) ;
  ttcul = mtime * (1 + utol) ;
  if (opal == 0) opal = ttcul ;
  last = 0 ;
  for (i = 0 ; i < models ; i++)     /* assign models to positions */
```

```
 { product [last] = i ;
  for (j = last + 1 ; j < last + batch_size [i] / bcd ; j++)
   product [j] = i ;
  last += batch_size [i] / bcd ;
 }
 return ;
}

float norm_dist (z)
float z ;
{
 float pr ;
 int zint ;
 if (z >= 3.5) return 1 ;
 if (z <= -3.5) return 0 ;
 zint = z * 100 ;
 if (z < 0) pr = 0.5 - normtabl [zint] ;
 else pr = 0.5 + normtabl [zint] ;
 return pr ;
}

float inccost (el,model)
int el ;
{
 int i ;
 float cost = 0 ;
 for (i = el+1 ; i <= last_el (statn (el)) ; i++)
 if (wk_el_avg [mlist [i][model]][model] != 0)
  cost += inc_cost [mlist [i][model]] ;
 for (i = last_el (statn (el)) + 1 ; i <= elements ; i++)
  if (prec [mlist[el][model]][mlist [i][model]] &&
   wk_el_avg [mlist [i][model]][model] != 0)
  cost += inc_cost [mlist [i][model]] ;
 return cost ;
}

find_mlist ()
{
 int i , j , first , second , model;
        /* find order within stations */
 for (model = 0 ; model < models ; model++)
 { for (i = 0 ; i <= elements ; mlist [i][model] = list [i] , i++) ;
  for (first = 1 ; first <= elements - 1 ; first++)
  {
   for (second = first+1 ; second <= last_el (statn (first)) ; second++)
   {
        i = mlist [first][model] ;
        j = mlist [second][model] ;
        if (inccost (first,model) < inccost (second,model) && !prec [i][j])
        { mlist [first][model] = j ;
          mlist [second][model] = i ;
```

```
        }
     }
    }
   }
  return ;
}

float prob_dist (dd, el, x , model)
int el , model;
float dd , x ;
{
  float m , s2 , z ;
  int i , st ;
  st = statn (el) ;
  m = d [st][part-1] ;
  if (d [st-1][part] > m) m = d [st-1][part] ;
  s2 = pow ((fmin (opal-mtime-m,m)) / 3 , 2) ;
  for (i = first_el (statn (el)) ; i< =el ; i++)
  { if (in_list (listh , i))
     m += wk_el_avg [mlist[i][cur_mod]][cur_mod] ;
     s2 += pow (wk_el_std [mlist[i][cur_mod]][cur_mod] , 2) ;
  }
  if (x= =m) return 0.5 ;
  if (s2 = = 0)
  { if (x>m) return 1;
    return 0 ;
  }
  z = (x-m) / sqrt (s2) ;
  return norm_dist (z) ;
}

float hist_prob (dd , k, model)
int k , model ;
float dd ;
{
  int i , f , j , l , st , lst , found , last_found ;
  float prob = 1 , pd , dd1 ;
  if (k< =0) return 0 ;
  lst = listh [k] ;
  for (st = 1 ; st < = statn (lst) ; st++)
  {
    dd1 = d [st][part-1] ;
    if (d [st-1][part] > dd1) dd1 = d [st-1][part] ;
    found = 0 ;
    f = first_el (st) ;
    i = last_el (st) + 1 ;
    if (i>lst) i = lst ;
    found = 0 ;
    while (i > first_el (st)  && found = = 0)
    { i-- ;
      if (in_list (listh,i))
```

```
    { found = 1 ;
        prob = prob * prob_dist (dd1 , i , opal , model) ;
    }
   }
   if (found != 0) i++ ;
   found = 1 ;
   j = i ;
   while (j <= last_el (st) && found != 0 && j < lst && st != statn (lst))
   { found = 0 ;
    for (l=1 ; l<=i-1 ; l++)
        if (!in_list (listh , l) && prec [mlist[l][model]][mlist[j][model]])
         found++ ;
    j++ ;
   }
   if (found == 0)
   { j-- ;
    prob = prob * (1 - prob_dist (dd1 , j , opal , model)) ;
   }
   if (fabs (prob) <= 0.000001) return 0 ;
  }
  prob = prob * (1 - prob_dist (dd1 , lst , opal , model)) ;
  return prob ;
}

find_cost ()
{
 int el , station , gg , model , i , val ;
 long history , hist ;
 float dd , inc_prob , f1 , f2 , pr_hist , cost = 0 ;
 if (list [0] == 0) return ;
 eval_solutions++ ;
 for (station = 0 ; station <= listg [0] ; d[station][0] = 0 , station++) ;
 for (part = 0 ; part <= products / bcd ; d[0][part] = 0 , part++) ;
 for (station = 1 ; station <= listg [0] ; station++)
 { for (part = 1 ; part <= products / bcd ; part++)
   { cur_mod = product [sched [part-1][cur_sc]] ;
    dd = d [station-1][part] ;
    if (d [station][part-1] > dd) dd = d [station][part-1] ;
    for (gg = first_el (station) ; gg <= last_el (station) ; gg++)
    {
        for (history = 1 ; history < pow (2 , gg) ; history += 2)
        { listh [0] = 0 ; hist = history ; val = 1 ;
          for (el = 1 ; el <= gg && val ; el++)
            if (hist - pow (2,gg-el) >= 0 )
            { listh [++listh [0]] = el ;
              hist -= pow (2,gg-el) ;
              val = valid_list (listh,cur_mod) ;
            }
          if (val)
          { inc_prob = hist_prob (dd, listh [0] , cur_mod) ;
            if (inc_prob != 0)
```

```
            {
             if (inc_prob != 0) cost += inc_prob * inccost (gg,cur_mod) ;
            }
           }
          }
          listh [0] = 0 ;
          for (i=1 ; i<listg[0] ; listh [++listh [0]] = i, i++) ;
          f1 = prob_dist (dd,last_el(station),mtime,cur_mod) ;
          f2 = prob_dist (dd,last_el(station),opal,cur_mod) ;
          d [station][part] = (opal-mtime) * (1 - (f1+f2)/2) ;
    }
   }
  }
  cost = labor_cost * listg [0] + bcd * cost ;
  gotoxy (5,8) ; printf (" Total Cost for previous combination %f ",cost) ;
  print_sol (cur_sc,cost) ;
  if (cost < best_c [low_best])
  {
  best_c [low_best] = cost ;
  copy_best () ;
  }
  return ;
}

int valid_list (listq,model)
int listq [] , model ;
{
  int el , i , j , found ;
  if (listq [0] == 0) return 1 ;
  el = listq [0] ;
  for (i = 0 ; i <= elements ; i++)
   if (!in_list (listq,i) && prec [list[i]][mlist[listq[el]][model]])
    return 0;
   for (i = first_el (statn (listq[el])) ; i < el ; i++)
    if (!in_list (listq , i))
    { found = 0 ;
         for (j = 1 ; j < i ; j++)
          if (!in_list (listq,j) && prec [list[j]][mlist[listq[i]][model]])
           found++;
         if (!found) return 0 ;
    }
  return 1 ;
}

                                        /** PRINT THE BEST BALANCE **/
print_best ()  /* print the best overall balance in list
                                        and listg; calculate station times and
                                        delta values */
{
  int i , j , k , stat , start1 ;
  float stime1 , delt1 [MAXMODS] , deltt , tt, deltot , ttot ;
```

```
  last_sol = imin (eval_solutions , MAXSOLS) ;
  fprintf (fout , "\n\nThe %d best combinations of balances ", last_sol) ;
  fprintf (fout , "and sequences out of %d investigated ", best_found) ;
  fprintf(fout,"\nin terms of lowest total expected cost are : \n\n") ;
  for (k = 0 ; k < last_sol ; k++)
  { stime1 = 0 ;
    start1 = 0 ;
    deltot = 0 ;
    ttot = 0 ;
    fprintf (fout,"Solution %d" , k + 1) ;
    for (stat = 0 ; best_listg [stat + 1][k] != 0 ; stat ++)
    { deltt = 0 ;
      stime1 = 0 ;
      for (j=0 ; j<models ; j++) delt1 [j] = 0 ;
          for (i = start1 ; i < best_listg [stat + 1][k] ; i ++)
      { stime1 += eltime [best_list [i+1][k]] ;
          for (j=0 ; j<models ; j++)
            delt1 [j] += wk_el_avg [best_list [i+1][k]] [j] ;
            tt = fabs (stime1 - mtime) ;
      }
      start1 = best_listg [stat + 1][k] ;
      for (j=0 ; j<models ; j++)
          deltt += fabs (model_delta [j] - batch_size [j] * delt1 [j]) ;
      deltot += deltt ;
      ttot += tt ;
    }
    copy_from_best (k) ;
    find_mlist ;
    for (j=0 ; j<models ; j++)
    { fprintf (fout,"\n\nModel %d: " , j + 1) ;
      start1 = 0 ;
      for (stat = 0 ; best_listg [stat + 1][k] != 0 ; stat ++)
      {
          fprintf (fout,"\nStation %d: " , stat + 1) ;
            for (i = start1 ; i < best_listg [stat + 1][k] ; i ++)
              fprintf (fout,"%3d" , mlist [i+1][j] + 1) ;
          start1 = best_listg [stat + 1][k] ;
      }
    }
    fprintf (fout,"\n\nWith a Delta value of %7.2f" , deltot) ;
    fprintf (fout," and a total difference of %7.2f\n\n" , ttot) ;
    fprintf (fout,"\nSequence of models: ") ;
    for (i=1 ; i<=best_sched[0][k] ; i++)
      fprintf (fout, "%2d " , product [best_sched [i][k]] + 1) ;
    fprintf (fout," (repeated %d times) \n\n" , bcd) ;
    fprintf (fout,"Total Expected Cost %f \n\n\n\n" , best_c[k]) ;
  }
  return ;
}

print_sol (cur_sc,cost)  /* print the best overall balance in list
```

```
                                        and listg; calculate station times and
                                        delta values */
int cur_sc ;
float cost ;
{
 int i , j , k , stat , start1 ;
 float stime1 , delt1 [MAXMODS] , deltt , tt, deltot , ttot ;
  stime1 = 0 ;
  start1 = 0 ;
  deltot = 0 ;
  ttot = 0 ;
  fprintf (fout,"Solution %d" , eval_solutions) ;
  for (stat = 0 ; listg [stat + 1] != 0 ; stat ++)
  { deltt = 0 ;
   stime1 = 0 ;
   for (j=0 ; j<models ; j++) delt1 [j] = 0 ;
        for (i = start1 ; i < list [stat + 1] ; i ++)
   { stime1 += eltime [list [i+1]] ;
        for (j=0 ; j<models ; j++)
          delt1 [j] += wk_el_avg [list [i+1]] [j] ;
   }
   start1 = listg [stat + 1] ;
   for (j=0 ; j<models ; j++)
        deltt += fabs (model_delta [j] - batch_size [j] * delt1 [j]) ;
   deltot += deltt ;
  }
  for (j=0 ; j<models ; j++)
  { fprintf (fout,"\n\nModel %d: " , j + 1) ;
   start1 = 0 ;
   for (stat = 0 ; listg [stat + 1] != 0 ; stat ++)
   {
        fprintf (fout,"\nStation %d: " , stat + 1) ;
         for (i = start1 ; i < listg [stat + 1] ; i ++)
           fprintf (fout,"%3d" , mlist [i+1][j] + 1) ;
        start1 = listg [stat + 1] ;
   }
  }
  fprintf (fout,"\n\nWith a Delta value of %7.2f" , deltot) ;
  fprintf (fout,"\nSequence of models: ") ;
  for (i=1 ; i<= sched[0][cur_sc] ; i++)
   fprintf (fout, "%2d " , product [sched [i][cur_sc]] + 1) ;
  fprintf (fout," (repeated %d times) \n\n" , bcd) ;
  fprintf (fout,"Total Expected Cost %f \n\n\n\n" , cost) ;

 return ;
}

load_schedules ()
{
 int i ;
 FILE *ftmp ;
```

```
 ftmp = fopen ("seq.tmp" , "r") ;
 scheds = 0 ;
 while (!feof (ftmp))
 { fscanf (ftmp, "%d " , &sched [0][scheds]) ;
  for (i = 1 ; i < = sched [0][scheds] ; i++)
   fscanf (ftmp, "%d " , &sched [i][scheds]) ;
  scheds++ ;
 }
}

int statn (n)                    /* station where nth element is assigned */
int n ;
{
 int k = 1 ;
 while (first_el (k) < = n && k< =listg[0]) k++ ;       /**** 2nd was < */
 return k-1 ;
}

int first_el (k)   /* first (lowest) element assigned to station k */
int k ;
{
 if (k = = 0) return 0 ;
 if (k = = 1) return 1 ;
 if (k > listg [0]) return elements+1 ;
 return listg [k-1] + 1 ;
}

int last_el (k)    /* last (highest) element assigned to station k */
int k ;
{
 if (k = = 0) return 0 ;
 if (k > listg [0]) return 0 ;
 return listg [k] ;
}

copy_best ()  /*  */
{
 int i ;
 best_found++ ;
 for (i=0 ; i<list [0] + 1 ; i++) best_list [i][low_best] = list [i] ;
 for (i=0 ; i<stations+2 ; i++) best_listg [i][low_best] = listg [i] ;
 for (i=0 ; i<sched[0][cur_sc] ; i++)
  best_sched [i][low_best] = sched [i][cur_sc] ;
 find_new_low () ;
 return ;
}

copy_from_best (l)  /*  */
int l ;
{
 int i ;
```

```
 for (i=0 ; i< =best_list [0][l] ; i++) list [i] = best_list [i][l] ;
 for (i=0 ; i< =best_listg [0][l] ; i++)
  listg [i] = best_listg [i][l] ;
 return ;
}

init_best ()
{
 int i , j ;
 for (i = 0 ; i < MAXSOLS ; i++)
 { best_c [i] = big_c ;
  best_list [0][i] = 0 ;
  for (j = 0 ; j < stations+2 ; j++)
   best_listg [j][i] = 0 ;
  best_sched [0][i] = 0 ;
 }
 low_best = 0 ;
 return ;
}

copy_vector (a1 , a2 , n)  /* put a copy of vector a1 to a2 of size n */
int a1 [] , a2 [] , n ;
{
 int i ;
 for (i=0 ; i<n ; i++)
  a2 [i] = a1 [i] ;
}

find_new_low ()
{
 int j ;
 low_best = 0 ;
 for (j = 1 ; j < solutions ; j++)
  if (best_c [j] > best_c [low_best]) low_best = j ;
}

int el_precedes (j)    /* true if element j has precedents */
int j ;            /* that are not part of the list */
{
 int i ;
 for (i=0 ; i<j ; i++)
  if ( (!in_list (list , i)) && prec [i] [j] ) return 1 ;
 return 0 ;
}

copy_list (a1 , a2)       /* put a copy of list a1 to list a2 */
int a1 [] , a2 [] ;
{
 int i ;
 for (i=0 ; i<a1 [0] + 1 ; i++) a2 [i] = a1 [i] ;
 return ;
```

```
}

int in_list (listq , k)          /* true if element k is part of list */
int k , listq [];
{
 int i ;
 for (i=1 ; i < = listq [0] ; i++)
 if (listq [i] = = k) return 1 ;     /*  return i + 1 ??? */
 return 0 ;
}

int imax (i , j)              /* maximum of integers i , j */
int i , j ;
{
 if (i > j) return i ;
 return j ;
}

int fmax (a , b)              /* maximum of reals i , j */
float a , b ;
{
 if (a > b) return a ;
 return b ;
}

int fmin (a , b)              /* minimum of reals i , j */
float a , b ;
{
 if (a < b) return a ;
 return b ;
}

int imin (i , j)              /* maximum of integers i , j */
int i , j ;
{
 if (i < j) return i ;
 return j ;
}
int sum (vector , n)                    /* sum of n first elements of vector */
int vector [] , n ;
{
int i , s ;
for (i = 0 , s = 0 ; i < n ; s + = vector [i] , i++) ;
return s;
}

void prtime ()                /* print the current date and time */
{
 time_t tnow ;
 time (&tnow) ;
 fprintf (fout,ctime (&tnow)) ;
```

```
}

                                        /** MAIN PROGRAM **/
main (int argc , char *argv [])
{
 int j ;
 float stime ;
 FILE *ftemp ;
 clrscr () ;
 if (argc != 4)
 { printf ("The cost evaluating program requires 3 arguments! \n") ;
  return ;
 }
 filein = argv [1] ;
 filerun = argv [2] ;
 fileout = argv [3] ;
 input_data () ;
 fout = fopen (fileout , "a") ;
 fprintf (fout,"\n\nStarting time for Cost Evaluation ") ;
 prtime () ;
 load_schedules () ;
 prep_data () ;
 printf ("Thinking...") ;
 ftemp = fopen ("bal.tmp" , "r") ;
 list [0] = 1 ;
 eval_solutions = 0 ;
 saved_bal = 0 ;
 while (!feof (ftemp) && list [0] != 0)
 { for (j = 0 ; j < elements + 1 ; j++)
    fscanf (ftemp , "%d " , &list [j]) ;
  for (j = 0 ; j < stations + 2 ; j++)
    fscanf (ftemp , "%d " , &listg [j]) ;
  find_mlist () ;
  scheds = 10 ;
  saved_bal++ ;
  gotoxy (5 , 4) ; printf ("Saved Balance %3d " , saved_bal) ;
  for (cur_sc = 0 ; cur_sc < scheds ; cur_sc++)
  { gotoxy (35,4);printf ("Schedule %3d " , cur_sc+1) ;
   find_cost (cur_sc) ;
  }
  if (saved_bal == 10) list[0] = 0 ;
 }
 fclose (ftemp) ;
 print_best () ;
 gotoxy (1 , 15) ;
 printf ("The results have been saved in file %s\n" , fileout) ;
 fprintf (fout,"End Time ") ;
 prtime () ;
 fclose (fout) ;
}
```

VITA

John Pantouvanos was born in Athens, Greece, on the 15th of October 1966. He attended a 5 year engineering program in Production Engineering and Management at the Technical University of Crete, in Greece. He obtained his Diploma in February 1990. He enrolled at Virginia Tech in August 1990 for a MS degree in Industrial Engineering and took his final examination in January 1992.

His research and professional interests include Manufacturing Automation, Applied Operations Research, and Computer Science.

John P. Pantouvanos