Models and Algorithms for a Flexible Manufacturing System

by

Rajendra Desai

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 Engineering and Operations Research

APPROVED:

Dr. Hanif D. Sherali, Chairman

Dr. Subhash C.Sarin

Dr. C. C. Wang

June, 1987

Blacksburg, Virginia

Models and Algorithms for a Flexible Manufacturing System

by

Rajendra Desai

Dr. Hanif D. Sherali, Chairman

Industrial Engineering and Operations Research

(ABSTRACT)

This thesis considers a Flexible Manufacturing System (FMS) comprised of programmable machine tools which are capable of performing multiple operations and which are interconnected by computer-controlled automated material handling equipment. The specific problem addressed is a job-shop type of situation in which at least a given number of each type of job needs to be performed on a given set of machines. The flexibility in the system arises in the form that each job can be performed in a variety of ways with each possible manner of performing it, called an Alternate Routing Combination (ARC), being defined by specifying the number of operations needed and the associated machine sequence. The problem is to select a set of jobs and their associated ARCs to be performed, and schedule their operations on the machines so as to optimize various objectives such as minimizing makespan or maximizing machine utilization, or minimizing total flowtime. This problem is mathematically modeled, and heuristic algorithms are presented along with computational results for the case of minimizing the makespan.

Acknowledgements

My sincere gratitude to Dr. Hanif Sherali for all the time and effort that he has devoted to my learning experince. Through his patience, dedication and wonderful teaching style, I have learnt to appreciate the concepts involved in the solution of complex problems. Without his help and guidance this thesis could not have been completed.

I am grateful to Dr. Subhash Sarin and Dr. C. C. Wang for all the time and assistance that they have so willingly and enthusiastically given me whenever I have required it. I would also like to thank them for their many helpful suggestions towards the completion of this thesis.

To my wife Ashu and my son Viren

Table of Contents

1. Introduction. 1

2. Literature Review 4

3. Mathematical Model 10

4. Selection of a Job-Arc Combination. 14

4.1. The Lagrangean relaxation approach for Heuristic 1. 14

4.2. An enumeration approach to job-arc selection for Heuristic 2. 25

4.3. Initial job - ARC selection routine for Heuristics 1 and 2. 26

5. Heuristic Scheduling Routine. 29

6. Summary of the Proposed Heuristics. 32

A summary of Heuristic 1 : 32

A summary of Heuristic 2 : 33

7. Computational Experience 35

8. Conclusions. 48

9. References. 50

Appendix A. Objective functions for the MIP. 58

1. Introduction.

The problem considered in this thesis is one that arises in the following Flexible Manufacturing System (FMS). The system consists of a set of N jobs and M machines. The N jobs are classified into mutually exclusive and collectively exhaustive sets N_1 ,..., N_D , with the stipulation being that at least α_d jobs must be performed from the set N_d . Each job $j = 1, \ldots, N$ can be performed in a variety of ways (a sequence of operations). Each manner of performing it is called an ARC (Alternate Routing Combination). An ARC is a sequence of operations required to complete a job, with the associated machines and the operation times for each operation specified. If a job is performed, exactly one ARC associated with it must be selected. Data on the number of possible ARCs for each job, the number of operations in each ARC, processing times and machine requirement for each operation are available. No preemption is allowed once a machine begins processing a job, and we assume that the transit and setup times between operations are negligible. The problem is to select a set of jobs to be performed along

with their associated ARCs, and schedule their operations on the machines so as to minimize the makespan.

The problem of determining a parts mix and scheduling in a FMS has been investigated by a number of researchers. A variety of approaches have been proposed based on ideas from Network of Queues, Mathematical Programming, Simulation, Petri Nets and Artificial Intelligence. Most of these combine heuristic ideas with exact methods. A review of these approaches is given in Chapter 2.

In this effort, two approaches are investigated. The first method, which we call Heuristic 1, uses a Mixed Integer Programming (MIP) model. The solution to the MIP uses a combination of Integer Programming techniques and heuristic methods to select a good set of job-arcs. The second method, called Heuristic 2, uses a quicker and less formalized enumeration type of method to select a good job-arc mix. It uses enumeration and employs heuristic ideas to initially select a job-arc mix and draw up a schedule for a given job-arc mix. Both these heuristics employ a specialized job-shop heuristic to draw up a final schedule.

The objective in our study is the minimization of makespan. Minimization of the makespan indirectly accounts for other objectives like machine utilization which are important in a Flexible Manufacturing System because of the high cost of the machines. However, other objectives can be easily incorporated with minor changes in the solution methods. Hence, we may consider minimization of the total idle time for the jobs, the total machine idle time (which is simply the difference between the makespan times the number of machines and the total processing time of all the jobs), or minimizing the maximum machine idle time (which is the same as maximizing the minimum machine

utilization). The mathematical formulation of some of these objective functions is given in the Appendix.

While Heuristic 1 met with limited success, Heuristic 2, though enumeration based, was very successful. This was primarily due to the effective heuristics which reduced the enumeration tree by cutting off non-promising solutions. Moreover, the specialized scheduling heuristic used at each node was very fast. The success of a method was measured in terms of the solution quality and the computational requirements. For smaller sized problems, an optimal makespan was obtained with the use of the MIP package MPSX. For larger problems the MPSX run time exceeded a few hours. Hence, it was prohibitively expensive to obtain an optimal solution. In such cases, the two heuristics were compared only on a relative basis.

2. Literature Review

A Flexible Manufacturing System has been defined as a production system consisting of programmable machine tools, which are capable of performing a number of different operations, and are interconnected by computer controlled material handling equipment. The term "flexible" refers to the system's ability 1) to perform a given set of operations on a part by using alternate routings through the machines and 2) to simultaneously work on several different part types. This results in a reduction in the work in process inventory and the set up time while changing from the production of one part type to another. A more detailed description of Flexible Manufacturing systems can be found in Sarin [1984], Kalkunte et al [1986], Barash [1982], Dupont-Gatelmand [1982] and the FMS hand book [1983].

The problem considered in this effort is one aspect of the design and operation of a FMS, namely, the selection of a parts mix and the scheduling of these parts to optimize a given objective.

Different objective functions have been used by researchers. Of these the most common ones are load balancing over the machines, minimization of part tardiness, maximization of the throughput rate and minimization of the makespan. Since the machines involved in an FMS are very expensive, maintaining a high level of machine utilization is often important for overall system effectiveness.

A number of different approaches have been proposed to deal with some or all of the aspects of the design and operation of a FMS. These have been categorized in Sarin [1984]. Five categories have been outlined : a) Network of Queues b) Mathematical programming c) Simulation d) Petri Nets and e) Artificial Intelligence based methods.

Of these, Simulation and Network of Queues approaches have been by far the most effective. Petri Nets and Artificial Intelligence methods are relatively new methods. Mathematical programming methods are very time consuming and have met with limited success.

A Network of Queues approach analyses the aggregate flow of pieces through the system rather than the movement of individual pieces. Machines are represented by nodes and the arcs represent the different routes for performing operations on the parts. This approach has been investigated by Kinemia [1979], who has used a Dantzig Wolfe decomposition on a derived linear program. The model used was a nonlinear program which was converted into a sequence of linearly constrained problems using an augmented Lagrangean method of Hestenes [1969] and Powell [1968]. The analysis was restricted to a "closed" network of queues, which means that the number of jobs in the system is constant (a new job enters the system only when an existing job leaves upon completion). A closed network of queues yields good results - this has been shown

theoretically (Suri [1983]) and empirically (Solberg [1982]) although its assumption of probabilistic service times is unrealistic.

Simulation has been by far the most widely used approach for FMS design and analysis. Various researchers (Hutchinson [1977], Mayer [1976], El Maghraby [1982], Phillips [1977]) have used simulation models to investigate different aspects of the system's operation and material handling. However, the drawback here is the excessive time and effort required in the developement and execution of these models. An improvement over this approach is the theoretical perturbation analyses of a single simulation run. (Ho et. al. [1979], [1983]). However, this approach is limited to small perturbations of the single simulation conditions.

Petri Nets (Peterson [1981]) are a compact representation of events in a system using nodes and arcs. This is a relatively new area and approaches have been outlined in Dubois and Stecke [1983], Narahari and Vishwanathan [1984], Nof et. al. [1980].

Artificial Intelligence methods use predicate logic and resolution. Several heuristic ideas are used to help the decision making under dynamic conditions. This is also a new approach in FMS design and is being investigated by various researchers. (Bullers et. al. [1980], Barr and Feigenbaum [1980]).

Mathematical Programming has been used for FMS design in the presence of failure prone machines (Hildebrant et. al. [1980], parts routing (Kimemia and Gershwin [1979]) and for machine loading (Stecke [1983], Kusiak [1983]). The disadvantage of this approach is in the excessive time required for the exact solution of the resulting models. A review and description of various mathematical models can be found in Buzacott [1980] and Wilhelm [1983].

Jaikumar et.al. [1984] have used a stochastic mathematical program which maximizes the total contribution of a mix of products, processes and programs within a manufacturing portfolio and provides the capability of sensitivity analyses. Kusiak [1984] has used a formulation based on clustering analysis (Anderberg [1973], Everitt [1980]) for the part family selection problem. The objective function used is to minimize the total distance between parts of different families. The number of part families is a parameter in the model. He gives another formulation which is not mathematically rigorous and the number of part families is a result of clustering. This is called the matrix formulation, and clustering involves moving around rows and columns so that part families are visible.

The work station loading problem is one of assigning tools, operations, and jobs to work stations in order to minimize the total number of job to work station assignments while balancing the work load per station. Ammons et. al. [1984] have investigated this problem by formulating it as a linear discrete optimization problem. The objective function explicitly considers the dual criteria of minimizing the total number of work station visits by jobs and maintaining a load balance on the work stations. They outline three algorithms with heuristic procedures imbedded to handle three different requirements. The work station balance criterion assigns operations in order of decreasing total processing times to the work station with the minimum work load at each step. To minimize work station crossings clustering analysis is used, where operations are grouped into clusters based on a similarity coefficient which describes the strength of the relationship between each pair of operations. To balance work station load and to minimize the number of crossings a combination of the previous two heuristics is used. Test problems with 31 jobs, 20.32 average number of operations per

job, 3 work stations and 30 tools per work station were used and good results were obtained with a small amount of computational time.

Stecke et.al. [1984] have formulated the loading problem as a nonlinear integer programming problem, the nonlinearity arising from the consideration of assigning several operations, having some common tools, to the same machine. Two approaches were attempted, the first involving a linearization of the problem with the resulting linear integer program solved using a standard MIP code. The second method tries to solve the nonlinear program directly, using an efficient branch and bound procedure which solves a relaxed assignment problem at each node, checks the solution for feasibility, and modifies the assignment if necessary to correct violations of constraints.

Lin et.al. [1984] have used heuristic procedures to balance the work load on each machine and to minimize the number of late jobs. These have been used in conjunction with the global dispatching rule of assigning the next operation to the machine which has the least work waiting.

Chang [1984] has used a two-phase approximate method for quasi-real time scheduling. It is quasi-real time because a new schedule is not necessarily drawn up after each event occurence. The specific problem considered by Chang is different from the FMS system considered here, in that he does not have the additional requirement of selecting ARC's for the jobs. His algorithm combines a schedule generation algorithm with a 0-1 optimization algorithm. Heuristics are used to select the jobs to be scheduled.

In this effort, the objective used is the minimization of makespan, since a number of objectives can be indirectly handled by optimizing this function. The model used is a linear mixed integer program. It is similar to Chang's approach in that it uses mixed

integer programming optimization techniques with heuristic ideas. However, we have used a more specialized schedule generation and scheduling algorithm. While Chang has used a specialized branch and filtering algorithm to select schedules for each job to optimize a specific criterion, we have used a subgradient based scheme (Heuristic 1) and an enumeration scheme (Heuristic 2). Chang has generated his schedules using a reduced enumeration algorithm developed by Chang and Sullivan [1984] which employs dispatching rules. Our schedules are generated using an active schedule generation scheme with a specialized heuristic routine to iteratively select the next operation-machine combination using results from Gere [1966]. Details on this are given in Chapter 4.

3. Mathematical Model

The MIP model developed uses a planning horizon T, which is discretized into $t = 1, \ldots, T$ intervals. The operations of a job are restricted to start and end at the discrete points in time defined by such intervals. This is done automatically by assuming that all processing times are an integral number of time intervals.

Define :

$j = 1, \ldots, N$: jobs to be performed.

$m = 1, \ldots, M$: machines in the system.

N_d : {j : job j is classified in job set d}, $d = 1, \ldots, D$

with $\sum_{d=1}^{D} |N_d| = N, N_l \cap N_t = \varphi, \quad \text{for} \quad l \neq t.$

α_d : specified minimum number of jobs to be performed from set N_d, $d = 1, \ldots, D$

A_j : {a: ARC a is to be considered for job j , $j = 1, \ldots, N$

n_j^a : number of operations in ARC $a \in A_j$ for job j , $j = 1, \ldots, N$

S : {p = (j,a) : p is a feasible job - ARC combination , $a \in A_j, j = 1, \ldots, N$ }

S_m : {(j,a,k) : operation k of ARC $a \in A_j$

of job j requires machine m } for $m = 1, \ldots, M$

S_m^{ja} : {k : operation k of ARC $a \in A_j$

of job j requires machine m } for $(j,a) \in S_m$, $m = 1, \ldots, M$

P_{jak} : processing time of operation k, $k \in a, a \in A_j, j = 1, \ldots; N$

$z_{ja} :=$ $\left\{ \begin{array}{l} 1 \text{ if ARC } a \in A_j \text{ of job j is selected, for } a \in A_j, j = 1, \ldots, N \\ 0 \quad \text{otherwise} \end{array} \right\}$

$x_{jakt} :=$ $\left\{ \begin{array}{l} 1 \text{ if operation k of ARC } a \in A_j \text{ of job j is being performed during the time interval t} \\ 0 \quad \text{otherwise} \end{array} \right.$

$k \in a, a \in A_j, j = 1, \ldots, N, t = 1, \ldots, T$

$\tau =$ makespan

Below we give the mathematical formulation of the problem with the objective of minimization of makespan. A formulation of alternate objectives is given in Appendix A.

minimize τ [3.1]

subject to

$$\tau \geq t \sum_{a \in A_j} x_{jan_j^a t} \qquad \text{for each} \quad j = 1, \ldots, N, \; t = 1, \ldots, T \qquad [3.2]$$

$$\sum_{a \in A_j} z_{ja} \leq 1 \qquad \text{for each} \quad j = 1, \ldots, N \qquad [3.3]$$

$$\sum_{J \in N_d} \sum_{a \in A_j} z_{ja} \geq \alpha_d \qquad \text{for each} \quad d = 1, \ldots, D \qquad [3.4]$$

$$\sum_{t=1}^{T} x_{jakt} = P_{jak} z_{ja} \qquad \text{for each} \quad k \in a, a \in A_j, j = 1, \ldots, N \qquad [3.5]$$

$$\sum_{r=1}^{k-1} \sum_{l=t}^{T} x_{jarl} + \sum_{l=t+P_{jak}}^{T} x_{jakl} \leq (z_{ja} - x_{jakt}) \sum_{r=1}^{k} P_{jar}$$

$$\text{for each} \quad k \in a, a \in A_j, j = 1, \ldots, N, t = 1, \ldots, T \qquad [3.6]$$

$$\sum_{(j,a,k) \in S_m} x_{jakt} \leq 1 \qquad \text{for each} \quad t = 1, \ldots, T, m = 1, \ldots, M \qquad [3.7]$$

x_{jakt} binary for each $k \in a, a \in A_j, j = 1, \ldots, N, t = 1, \ldots, T$

z_{ja} binary for each $a \in A_j, j = 1, \ldots, N$ [3.8]

$x_{jakt} = 0$ for $t = 0, \ldots, \overline{T}_{jak}$ and for $t = T + 1 - \hat{T}_{jak}, \ldots, T + 1$

$$\text{for each} \quad k \in a, a \in A_j, j = 1, \ldots, N, \qquad [3.9]$$

where $\overline{T}_{jak} = \sum_{r=1}^{k-1} P_{jar}$ and $\hat{T}_{jak} = \sum_{r=k+1}^{n_j^a} P_{jar}$, with undefined sums being zero

In the above MIP, constraint [3.2] ensures that the makespan τ is greater than the finish time of the last operation of the ARC selected for each job j. Hence, with the objective [3.1], it defines the makespan. Constraints [3.3] and [3.4] state that only one ARC may be selected for each job, while at least α_d jobs must be selected from the set N_d. Constraints [3.5] ensure that if a job - arc combination is selected, each of it's operations is allocated the number of time intervals required to complete its processing. The constraints [3.6] ensure that an operation of a selected job-arc combination will start only after all its preceding operations have been completed. Constraints [3.7] prevent multiple job assignments to the same machine in any time interval. Constraints [3.8] are the binary requirements on the decision variables. The constraints [3.9] tighten the relaxation by specifying the intervals over which the x variables must be set equal to zero.

The overall solution strategy for the MIP is as follows : Constraints [3.2] and [3.7] are dualized to obtain a Lagrangean dual with subproblems which are easy to solve. A conjugate subgradient type of algorithm is used to obtain a set of near optimal dual multipliers associated with the constraints. The primal solution obtained at the termination of the subgradient algorithm gives a good set of job-arc combinations. This set of job arc mix is used with a heuristic scheduling routine to draw up a good schedule. Details on this method and a quick heuristic method for selecting a job-arc combination are given in Chapter 4. The specialized initial job-arc selection routine and the scheduling heuristics used in both Heuristic 1 and Heuristic 2 are given in Sections 4.3 and Chapter 5, respectively.

4. Selection of a Job-Arc Combination.

Two methods are used in the selection of a good set of job-arc combinations. The first is based on a Lagrangean relaxation and subgradient optimization technique on the Mixed Integer Program outlined in the previous section. The second uses an enumeration technique cutting down the number of nodes in the enumeration tree by using some simple bounds and evaluating each job-arc mix at each node. The Lagrangean relaxation approach also results in some additional sequencing information which is used by the scheduling routine. The first method of job-arc selection is used in Heuristic 1 and the second is used in Heuristic 2.

4.1. The Lagrangean relaxation approach for Heuristic 1.

The mathematical model outlined in the previous section requires a prohibitive effort to solve exactly by any of the currently available Integer Programming algorithms for

problems of any reasonable size. In experiments on this model using the MIP package MPSX on the IBM 3084 main frame system, the largest problem which could be run had only 4 jobs with 3 arcs per job and 3 operations per arc. The length of the horizon for the completion of all the operations had to be limited to 6 time units. Further details on this are given in Section 7. Hence, a heuristic based on a Lagrangean relaxation is employed.

The Lagrangean dual of the MIP is obtained by dualizing constraints [3.2] and [3.7] using the multipliers β_{jt} for the constraint set [3.2] and δ_{tm} for the constraint set [3.7].

This gives the following dual problem :

$$\text{LD} : \quad \text{maximize} \quad \{\Theta(\beta, \delta): \sum_{j=1}^{N} \sum_{t=1}^{T} \beta_{jt} = 1, \quad \beta \geq 0, \ \delta \geq 0\} \qquad [4.1]$$

where

$$\Theta(\beta, \delta) = \text{minimum} \sum_{j=1}^{N} \sum_{t=1}^{T} \sum_{a \in A_j} \beta_{jt} t x_{jan_j^a t}$$

$$+ \sum_{t=1}^{T} \sum_{m=1}^{M} \delta_{tm} [\sum_{(j,a,k) \in S_m} x_{jakt} - 1] \qquad [4.2]$$

subject to

$$\sum_{a \in A_j} z_{ja} \leq 1 \qquad \text{for each} \quad j = 1, \ldots, N \qquad [4.3]$$

$$\sum_{j \in N_d} \sum_{a \in A_j} z_{ja} \geq \alpha_d \qquad \text{for each} \quad d = 1, \ldots, D \qquad [4.4]$$

$$\sum_{t=1}^{T} x_{jakt} = P_{jak} z_{ja} \qquad \text{for each} \quad (jak) \quad \text{combination.} \qquad [4.5]$$

$$\sum_{r=1}^{k-1} \sum_{l=t}^{T} x_{jarl} + \sum_{l=t+P_{jak}}^{T} x_{jakl} \leq (z_{ja} - x_{jakt}) \sum_{r=1}^{k} P_{jar}$$

$$\text{for each} \quad k \in a, a \in A_j, j = 1, \ldots, N, t = 1, \ldots, T \qquad [4.6]$$

$$x_{jakt} = 0 \quad \text{for } t = 0, \ldots, \overline{T}_{jak} \quad \text{and for } t = T + 1 - \hat{T}_{jak}, \ldots, T + 1 \quad \text{for each } (jak) \qquad [4.7]$$

$$x, z \quad \textit{binary} \qquad [4.8]$$

Note that the integrality property does not hold here, and so we achieve a tightening beyond the continuous relaxation. For convenience, let us denote the objective function [4.2], with obvious notation as

$$\Delta_{\beta\delta} + \quad \text{minimum} \quad \sum_{j=1}^{N} \sum_{a \in A_j} \sum_{k \in a} \sum_{t=1}^{T} C_{jakt}^{\beta\delta} x_{jakt} \qquad [4.9]$$

Now, the subproblem [4.2] - [4.8] may be rewritten as :

$$\Theta(\beta,\delta) = \Delta_{\beta\delta} + \left\{ \begin{array}{l} \text{minimum} \\ \text{z binary} \\ \text{[4.3], [4.4]} \end{array} \left\{ \begin{array}{l} \text{minimum} \sum_{j=1}^{N} \sum_{a \in A_j} \sum_{k \in a} \sum_{t=1}^{T} C_{jakt}^{\beta\delta} x_{jakt} \\ \text{x binary} \\ \text{[4.5], [4.6], [4.7]} \end{array} \right\} \right\} \qquad [4.10]$$

Note that the inner minimization problem is separable in (j,a). If we denote this problem by P(j,a), then for any fixed z, we get

$$P(j,a) \quad : F_{ja}(z_{ja}) = \text{minimize} \sum_{k \in a} \sum_{t=1}^{T} C_{jakt}^{\beta\delta} x_{jakt} \qquad [4.11]$$

subject to

$$\sum_{t=1}^{T} x_{jakt} = P_{jak} z_{ja} \qquad \text{for each} \quad k,$$

$$\sum_{r=1}^{k-1} \sum_{l=t}^{T} x_{jarl} + \sum_{l=t+P_{jak}}^{T} x_{jakl} \leq (z_{ja} - x_{jakt}) \sum_{r=1}^{k} P_{jar}$$

$$k \in a, t = 1, \ldots, T.$$

and [4.7] and [4.8] for each k, for the chosen (j,a).

Let $C_{ja}^{\beta\delta} = F_{ja}(1)$. Since, $F_{ja}(0) = 0$, we have

$$F_{ja}(z_{ja}) = z_{ja} C_{ja}^{\beta\delta} \qquad [4.12]$$

Consequently, from [4.10], [4.11], [4.12], we obtain

$$\Theta(\beta,\delta) = \Delta_{\beta\delta} + \text{minimum} \{ \sum_{j=1}^{N} \sum_{a \in A_j} C_{ja}^{\beta\delta} z_{ja} : \text{[4.3], [4.4], z binary} \} \qquad [4.13]$$

We now need to solve for $C_{ja}^{\beta\delta} = F_{ja}(1)$, and then show how to solve [4.13] efficiently.

Computation of $C_{ja}^{\beta\delta} = F_{ja}(1)$:

Toward this end, first consider the following result.

Proposition 1 : (z,x) is feasible to [4.11] if and only if

(a)

$x_{jakt} = z_{ja}$ for some P_{jak} contiguous values of t in the interval $\overline{T}_{jak} + 1, \ldots, T - \hat{T}_{jak}$

and are zero otherwise, for each $k \in a$,

(b) the operations of (j,a) are ordered, i.e.

$x_{japq} = z_{ja}$ implies $x_{ja(p-1)t} = 0$ for $t \geq q$ for each $p = 2, \ldots, n_j^a$

Proof :

If $z_{ja} = 0$, the result is trivial. Hence, suppose $z_{ja} = 1$, and assume that (z,x) satisfies [4.11]. From [4.11], if $x_{jakt} = 1$, then $x_{jarl} = 0$ for all $r = 1, \ldots, k-1, l = t, \ldots, T$ which ensures an ordering of the operations of $a \in A_j$ since operation (k-1) will never be done after operation k. Further, $x_{jakl} = 0$ for all $l = t + P_{jak}, \ldots, T$. Since, $z_{ja} = 1$, P_{jak} x's must be set equal to 1 for operation k in $a \in A_j$. Hence, examining these assertions for the first t for which $x_{jakt} = 1$, we get that the P_{jak} x's which are equal to 1 have to be contiguous.

Conversely, if the operations of $a \in A_j$ are ordered, then for any $x_{jakt} = 1$, all it's previous operations will be completed by time t. Further, if P_{jak} contiguous x's are set equal to 1, then $x_{jakl} = 0$ for all $l \geq (t + P_{jak})$. Hence, [4.11] will be satisfied. (Q.E.D.)

We now evaluate C_{ja}^{ps} via a Dynamic Programming procedure. The stages, states and decisions are defined as follows :

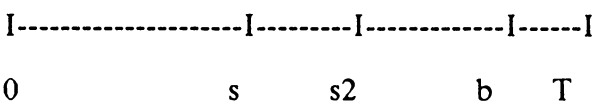
Stage (k) : one stage for each operation belonging to (j,a). Hence, each (j,a) will have n_j^a stages.

State (S_k) : defines a feasible starting time t for the consideration of the remaining operations in a backward recursion. State reduction is performed based on the set [4.7]. In this reduced state space there is a natural division into blocks over which the search can be limited as outlined below.

Decision variable (d(k,S)) : the point at which the P_{jak} consecutive values of operation k must be started. This is beyond the particular state value S at stage k.

State space reduction

The natural division of the reduced state space is as follows :



If the reduced state space of some operation k is [s,b], we can use a one pass procedure to optimize on the interval [s,b] taking the costs over P_{jak} x's at a time. This is done by starting at (s) or (s + 1) or ,.., or $(b - P_{jak})$.

Suppose, this results in an optimal start point for operation k at s2. Then this is obviously optimal for any state in the interval [s,s2]. Hence, our search can now be limited to the block [s2,b]. This results in substantially reducing the number of d(k,S) evaluations for each operation.

The Dynamic programming algorithm is a standard backward recursion procedure. Starting with the last operation n_j^a (stage) of each job-arc, we obtain the optimal $P_{jan_j^a}$ time slots for each of the state variable values over its reduced state space. This is repeated for operations (k - 1),..,1 with each stage decision taking into account the total effect of this decision on all its following stages (via the modification of the state variable by this stage).

The DP solution for each (j,a) gives us a time $\bar{t}$ for each operation k belonging to (j,a). Accordingly, let :

$$I_{jak} = \{\bar{t}, \bar{t} + 1, ..., \bar{t} + P_{jak} - 1\} \qquad [4.14]$$

The x solution is then given in terms of the z variables as follows :

$$x_{jakt} = \left\{ \begin{matrix} z_{ja} & \text{if} \quad t \in I_{jak} \\ 0 & \text{otherwise.} \end{matrix} \right\} \qquad [4.15]$$

Furthermore, this reduces the subproblem to the form [4.13], where

$$C_{ja}^{\beta\delta} = \sum_{k \in a} \sum_{t \in I_{jak}} C_{jakt}^{\beta\delta} \qquad [4.16]$$

The subproblem [4.13] is now easily solved as follows :

For each j, let a[j] be determined by :

$$C_{ja[j]}^{\beta\delta} = \underset{a \in A_j}{\text{minimum}} \{C_{ja}^{\beta\delta}\} \qquad [4.17]$$

Then, we obtain

$$\Theta(\beta, \delta) = \Delta_{\beta\delta} + \sum_{d=1}^{D} \{ \text{ sum of the } \alpha_d \text{ smallest values of } C_{ja[j]}^{\beta\delta} \text{ over } j \in N_d \} \quad [4.18]$$

The corresponding value of z_{ja} is given by

$$z_{ja} = \left\{ \begin{array}{ll} 1 & \text{if a = a[j] and } C_{ja[j]}^{\beta\delta} \text{ contributes to the sum in [4.18]} \\ 0 & \text{otherwise} \end{array} \right\} \quad [4.19]$$

Equations [4.14] and [4.19] give the subproblem solution. The overall Lagrangean optimization problem [4.1] is solved as follows :

Subgradient Algorithm

The concave Lagrangian dual objective function $\theta(.)$ is maximized over the dual space using a conjugate subgradient optimization scheme which combines the ideas of Camerini, Fratta and Maffioli [1975] and Bazaraa and Sherali [1981].

Let us denote the k^{th} iterate by π^k . Let $P_D(\pi) = \bar{\pi}$ be the projection of π onto the feasible space D, where D is given by

$$D = \{ \pi = (\beta, \delta) : \sum_{j=1}^{N} \sum_{t=1}^{T} \beta_{jt} = 1, \beta, \delta \geq 0 \}$$

Let ξ^k denote the subgradient of $\theta(.)$ at π^k . The components of ξ correspond to the coefficients of β and δ in the Lagrangean objective function.

INITIALIZING THE ALGORITHM

1. An initial feasible starting set $\pi^0 = (\beta^0, \delta^0)$ is needed to start the algorithm. (We used $\pi^0 = 0 -$ vector).

2. We need an upper bound on the value of $\theta(\pi^*)$ (This was obtained via a heuristic solution to the original problem).

3. We use the following parameters in the subgradient scheme.

   N = Maximum number of iterations.

   $\bar{v}$ = Number of consecutive failures before resetting to incumbent.

   $\bar{r}$ = Number of iterations at which we reset to the incumbent and recalculate the step size.

The typical values for these parameters are N = 300, $\bar{v}$ = 10 and $\bar{r}$ = 75.

All counters are set to 0 before beginning the solution procedure. Starting with the initial solution π^0 at k = 0, at any general iteration k, let $\theta(\pi^k)$ denote the solution of the Lagrangean problem given a particular dual vector π^k . This value is obtained by the Dynamic Programming procedure outlined in this section. This solution also yields a subgradient ξ^k of $\theta(.)$ at π^k , based on the coefficients of π in the objective function of the Lagrangean subproblem. If the norm of the subgradient $\|\xi^k\|$ is lesser than a preset value of ε, we stop with π^k as a near optimal solution of value $\theta(\pi^k)$.

If this is not the case, we check to see if $\theta(\pi^k)$ is greater than the present incumbent value. If so, the incumbent solution is reset to $\pi^* = \pi^k$ and $\theta^* = \theta(\pi^k)$. To compute the next point π^{k+1} we have to calculate a direction and a step length. The direction d^k is calculated by taking a combination of the subgradient and the direction at the previous point.

$$d^k = \xi^k + \alpha_k d^{k-1}$$

where

$$\alpha_k = \frac{-\Delta\, d^{k-1}\xi^k}{\|d^{k-1}\|^2} \qquad \text{if } d^{k-1}\xi^k < 0$$

$$\alpha_k = 0 \qquad \text{otherwise}$$

The step length is calculated based on Bazaraa and Sherali [1981], as

$$s = \left[\frac{\bar{\theta} - \theta(\pi^k)}{\|d^k\|^2}\right]\left[\frac{1 - 0.5(n-1)\bar{r}}{(N-\bar{r})}\right]$$

The step length is periodically updated and resetting to the incumbent is also periodically performed based on the counter thresholds $\bar{v}$ and $\bar{r}$ as in Bazaraa and Sherali [1981].

The vector might not be in the feasible region. Therefore the vector is finally projected onto the feasible region D and this gives the new iterate. The process now continues until either $\|\xi^k\| \leq \varepsilon$ or $k \geq N$

A further dual ascent can be achieved by starting with the current (β, δ) and solving the same type of dual problem as [4.1] - [4.8] , except now each job has $|A_j| = 1$ and $\alpha_d = |N_d|$, so that there is no optimization over the z's. However, this was not implemented because of the considerable computational expense. Once the z's have been fixed, the indices (j,a) get collapsed to essentially the index j, and the problem becomes :

minimize τ [4.20]

subject to :

$$\tau \geq t x_{jn_j t} \qquad j = 1, \ldots, N, t = 1, \ldots, T \qquad [4.21]$$

$$\sum_{t=1}^{T} x_{jkt} = P_{jk} \qquad \text{for all} \quad j, k \qquad [4.22]$$

$$\sum_{r=1}^{k-1} \sum_{l=t}^{T} x_{jrl} + \sum_{l=t+P_{jk}}^{T} x_{jkl} \leq (1 - x_{jkt}) \sum_{r=1}^{k} P_{jr} \quad \text{for all} \quad j,k,t = 1,\ldots,T - P_{jk} \qquad [4.23]$$

$$\sum_{(jk) \in S_m} x_{jkt} \leq 1 \qquad t = 1, \ldots, T, m = 1, \ldots, M \qquad [4.24]$$

$$x \qquad \text{binary} \qquad [4.25]$$

Now, we can dualize constraints [4.21] using the dual multipliers β_{jt} and constraints [4.24] using the multipliers δ_{tm} and follow the same procedure as before to acheive a further dual ascent.

Job - ARC selection and priority indexing.: Each iteration of the subgradient algorithm produces a new set of dual variables β and δ. These modify the costs $C^{\beta\delta}_{jakt}$ and at each iteration a set of z_{ja} is selected to be set equal to 1. These are the job - arc selections for that iteration. The scheduling routine of Chapter 5 is used to draw up a schedule for each such set. The schedule which gives the best makespan is stored along with the associated job - arc selections and the x values.

The scheduling routine used here requires a priority index based on the x solution obtained at each iteration of the subgradient algorithm. This is defined as follows :

$$I_{jk}(\text{ c.t }) = \sum_{t=1}^{\text{c.t. } + P_{jak}} x_{jakt}/P_{jak}$$

for each j, "a" corresponding to the arc selected for job j, and each k in this selected arc. Here c.t. is the time in the scheduling algorithm at which the operation k is ready to be scheduled. This index can be interpreted as follows : In every iteration, for operation k of the selected arc "a" of a job j , the Lagrangeàn relaxation solution results in P_{jak} of the x_{jakt} values set equal to 1. In the scheduling routine, if this operation is schedulable at time t_1 then one indication of its priority is taken to be the fraction of the total number of x's which have been set equal to 1 (by the Lagrangean solution) until time t_1. Of all the schedulable operations at time t_1 , the one with the highest value of $I_{jk}(t_1)$ is selected.

4.2. An enumeration approach to job-arc selection for Heuristic 2.

The enumeration approach enumerates (explicitly) all feasible job-ARC combinations. At each node, a complete set of feasible job-ARCs is drawn up and evaluated using lower bounds or a complete schedule. For each feasible job - ARC combination, lower bounds are used to reject the solution (if possible) before evaluation by the heuristic scheduling routine outlined in Chapter 5. The best makespan value is stored as the upper bound on the makespan. The enumeration tree is reduced considerably by deriving certain lower bounds on the makespan at each node. Three trivial lower bounds are used. The first simply sums up the operation times of each arc and takes the largest of these

as the lower bound. The second lower bound is the sum of all the operation times over the selected job-arcs divided by the number of machines. The third lower bound is a heuristic lower bound, based on a schedule drawn up for half the actual number of jobs to be scheduled. The heuristic scheduling routine is used for this purpose.

While these lower bounds are not very strong, they are adequate for our purposes, since the actual evaluation of a job-arc combination (using the heuristic scheduling routine) is not computationally expensive. Improvements on this approach can be made for handling very large problems by using tighter lower bounds. Additionally, an implicit enumeration may be conducted in which a lower bound on a partial set of enumerated job-arc combinations is used to eliminate all possible completions, hence curtailing the extent of enumeration. Also, some ARCs, and possibly some jobs may be eliminated prior to enumeration based on some heuristic considerations, in order to further curtail the extent of enumeration. The upper bound at node zero is obtained by a heuristic job-arc selection routine given in the following section.

4.3. Initial job - ARC selection routine for Heuristics 1 and 2.

The two heuristics Heuristic 1 and Heuristic 2 use a common initial job-arc selection routine, which is a quick procedure based on the magnitude of the operation times and the number of operations in an arc. This solution is used as an upper bound on the optimal solution in both Heuristic 1 and Heuristic 2.

The procedure operates as follows :

Initialize d = 1.

Start with set number N_1 and pick a job from this set for consideration.

Step a) Of the arcs belonging to this job find those arcs with a total processing time within 10 % of the minimum total arc processing time. Of these arcs pick the arc with the minimum number of operations and include this job-arc combination in the selected job-arcs list.

Step b) If this is not the last job in this set, then look at another job in this set which has not been considered so far, and return to step (a).

Step c) If the number of jobs selected from this set exceeds α_d for this set then go to step (e).

Step d) If there are any more sets not yet considered, then increment the set number by 1 and go to step (a). Else, stop.

Step e) Since the number of jobs selected exceeds α_d , we need to remove the excess number of jobs over α_d selected from this set. This is done as follows :

1) Order the jobs in increasing order of processing times
of the selected arcs.

2) Pick the α_d jobs with the smallest total processing time (for the arc selected for each such job), breaking ties using the smaller number of operations. Call this list NL.

3) Determine the maximum arc processing time from list NL.

4) From the previously selected jobs not included in this new list, determine those with arc processing times within 10% of the maximum arc processing time in NL. Put these jobs in a list called the left out jobs list or LOJ. Concatenate lists NL and LOJ, assuming that the jobs are ordered. Call this new list L. We need to throw $|LOJ|$ number of jobs out of the list L.

5) Start with r = 0 (r = number of jobs thrown out of list L thus far).

a) If r = $|LOJ|$ then go to (e), else pick the bottom job from L. Let this be job j with arc processing time t_j Select that job k from L which has $t_k \geq t_j/1.1$ and the largest

number of operations, breaking ties by picking the one with a larger arc processing time (possibly, k = j). Delete job k from L.

b) Increment r by 1 and return to 5.

5. Heuristic Scheduling Routine.

The scheduling routine discussed in this Chapter is used by both Heuristic 1 and Heuristic 2. This heuristic routine is based on generating an active schedule using a procedure proposed by Giffler and Thompson [1960] adapted by us as a heuristic procedure. The framework for generating active schedules is used as outlined in Giffler and Thompson, along with a specialized heuristic used for breaking ties between competing operations. The outline of the algorithm is given below :

Define :

PS_t = a partial schedule containing t scheduled operations.

(This defines stage t as the point in the process when some t total operations have been scheduled. Accordingly, let the "current time" c.t. = the partial solution makespan).

S_t = the set of schedulable operations at stage t, corresponding to a given PS_t . Each schedulable operation is identified by the imminent operation k_j of a job j, with a particular machine being associated with operation k_j of job j.

σ_{k_j} = the earliest time at which schedulable operation $k_j \in S_t$ can be started.

φ_{k_j} = the earliest time at which schedulable operation $k_j \in S_t$ can be completed. (This is = $\sigma_{k_j} + P_{ja k_j}$ where "a" is the selected ARC for job j).

$\overline{\varphi}_j$ = the (earliest) time at which job j can be completed via a non delay schedule of the unscheduled operations.

Further, we define the notion of critical jobs and priority indices below :

Critical Jobs

Given a partial schedule, consider the imminent schedulable operation as k_j of job j. Suppose that operation k_j is scheduled on its corresponding machine. For each job (including job j), construct independently, a nondelay schedule of its remaining operations, inserting operations on feasible gaps in machines, if possible, in order to determine the earliest finish time ($\overline{\varphi}_j$) of job j. If $\overline{\varphi}_j \geq$ [due date of job j] * (factor) then job j is said to be critical. Here, the due dates can be actual due dates for the jobs or some artificial due dates based on a tight upper bound on the makespan. (In experimental runs with artificial due dates, it was found that the scheduling routine performed well if these were set close to the optimal makespan value. Since the initial job-ARC makespan value was very good, in all our schedules this value was used as the artificial due date for the jobs. For the initial job-ARCs schedule, the artificial due date value was set at 1.5 times the sum of the operation times divided by the number of machines). The "factor" used in the critical job determination was fixed experimentally. A factor of 1.5 seemed to perform well on a number of test problems. Having done this, let N_{k_j} = the resulting number of critical jobs if operation k_j is scheduled next.

Priority indexing of the operations

The job to be scheduled is selected as the arg lex min { N_{k_j}, σ_{k_j}, P_{jak_j}, I_{jk_j}(c.t.) } (where k_j is a member of a subset of the schedulable operations at this stage, as noted in the overall algorithmic strategy below, and N_{k_j} is the number of critical jobs due to operation k_j being scheduled, P_{jak_j} is the operation time for the imminent operation k_j of job j corresponding to the selected arc, c.t. is the current time interval, and I_{jk} (.) is as defined in Section 4.1), (Here arg lex min { a_j, b_j, c_j} = that j which minimizes a_j, and among all such j's, it minimizes b_j , and among all such ties , minimizes c_j.)

Overall algorithmic strategy :

1) Let t = 0 and begin with PS_t empty and S_t as the set of all operations with no predecessors. (i.e. the first operation of each job corresponding to the selected ARCs).

2) Determine $\bar{\varphi} = \min \varphi_{k_j}$ over all $k_j \in S_t$ and the machine $\bar{m}$ on which $\bar{\varphi}$ is realized.

3) For each operation $k_j \in S_t$ that requires machine $\bar{m}$ and for which $\sigma_{k_j} < \bar{\varphi}$, use the above priority index to determine the operation to be added to the current partial solution PS_t as early as possible, creating a new partial solution PS_{t+1}. (Break persisting ties arbitrarily).

4) For the new partial schedule created, update the data set as follows :

a) Remove operation k_j from S_t.

b) Form S_{t+1} by adding the direct successor of operation k_j to S_t.

c) Increment t by one.

5) Return to step 2 with the current partial schedule and continue until a complete schedule is generated.

6. Summary of the Proposed Heuristics.

A summary of Heuristic 1 :

Step 1. Obtain an initial job - ARC set using the heuristic routine outlined in Section 4.3.

Step 2. Draw up a schedule for this job-ARC set , using the scheduling routine outlined in Chapter 5. This will be an upper bound on the makespan.

Step 3. Use the makespan obtained in Step 2 as an upper bound for the subgradient algorithm, which is used to solve the Lagrangean relaxation outlined in Section 4.1. Use the initial job-ARC solution to obtain an advanced dual start for the subgradient algorithm by determining the binding / non-binding constraints. An all zero dual start can also be used.

Step 3 (a). At each subgradient iteration, given a set of dual multipliers, solve the Lagrangean subproblem using a Dynamic Programming procedure as shown in Section 4.1. This leads to the selection of a set of job-ARCs.

Step 3 (b). Use the scheduling routine [Chapter 5] to draw up a schedule for the job-ARC set obtained in step 3 (a), using the x solution generated by the DP procedure for priority indexing. Store this set of job-ARCs, if the resulting makespan is better than the best makespan so far.

Step 3 (c). Determine a subgradient direction to move along and obtain a new set of feasible dual multipliers [see Section 4.1 - Subgradient algorithm]. Repeat step 3 for a certain maximum number of iterations. At the end of these iterations the best set of job-ARCs and the corresponding schedule are the solution obtained using Heuristic 1.

A summary of Heuristic 2 :

Step 1. Obtain an initial job - ARC set using the heuristic routine outlined in Section 4.3.

Step 2. Draw up a schedule for this job-ARC set , using the scheduling routine outlined in Chapter 5. This will be an upper bound on the makespan.

Step 3. Use an enumeration scheme to select a set of job_ARCs which has not been evaluated so far, ensuring the requirements of selecting α_d jobs from the set $N_d, d = 1, \ldots, D$ and 1 ARC per job.

Step 4. Calculate the lower bounds given below in the order in which they are described. If any of these is greater than the current upper bound, go to step 3. Else continue with the next lower bound calculation. If none of the lower bounds result in the rejection of this job-ARC set, then go to step 5.

lower bound 1 Sum up the operation times of each ARC and take the largest of these as lower bound 1.

lower bound 2 Sum up the operation times of all the ARCs selected and divide this sum by the total number of machines available to obtain lower bound 2.

lower bound 3 For each machine, sum up the operation times for the operations of the selected job-ARCs assignable to it and take the largest of these to obtain lower bound 3.

lower bound 4 Draw up a schedule using the scheduling routine of Chapter 5 using half the total number of selected jobs with their corresponding ARCs, to obtain a heuristic lower bound 4.

Step 5 Draw up a complete schedule and if the resulting makespan is better than the current upper bound, make this the new upper bound and store this solution. Go to step 3 and repeat until all job-ARC combinations have been considered. The solution corresponding to the final upper bound is the solution obtained using Heuristic 2.

7. Computational Experience

In this chapter, we present the computational results obtained with Heuristics 1 and 2. All experimentation was done on an IBM 3090 computer with the coding in Pascal. The first set of runs were made using the Integer Programming package of MPSX, to get an idea of the complexity of the model. The results summarized in Table 7.1 show the effect of changing the number of jobs, the number of ARCs (alternate routing combinations) per job, the number of operations per ARC and the planning horizon, on the run time. It is clear that problems with more than 4 jobs, 3 ARCs per job and 3 operations per ARC require a prohibitive amount of computer time. Hence, it is prohibitive to solve any real sized problems using a standard MIP package. The Linear Programming (LP) solution value indicates a large duality gap which is also apparent from the results of Heuristic 1.

In further experimentation, problems were generated to investigate the effects of varying the number of jobs (J), the maximum number of ARCs per job (A), the maximum

number of operations per ARC (K) and the total number of machines available (M). A uniform random number generator was used to generate data in the following manner : The number of jobs and the total number of machines available were input along with the upper limits A and K on the number of ARCs per job, and on the number of operations per ARC respectively. The actual number of ARCs for any job, operations per ARC, machine requirements and processing times were generated randomly within their prescribed limits.

Each operation of an ARC of a job was made to require exactly one machine, with all operations of the same ARC of a job requiring different machines.

The number of subsets D used is determined by a requirement to balance the jobs per set with the jobs being allocated to the sets one at a time. Hence, set 1 has jobs 1, (D + 1), (2D + 1) ... and set 2 has jobs 2, D + 2, ... and so on. The actual jobs α_d to be performed from a given subset N_d is randomly generated within it's limits.
($1 \leq \alpha_d \leq |N_d|$).

The scheduling horizon is determined as 1.2 times the makespan of the initial heuristic schedule described in Chapter 5. The motivation for this is to make the horizon as tight as possible to prevent the number of variables from getting too large. For a tighter horizon, a factor of 1.0 could have been used on the initial makespan. However, this results in some infeasible subproblems when the Dynamic Programming procedure is used for their solution. While the factor of 1.2 is not guaranteed to work for all problems, it was found to be satisfactory for all our test problems. (Whenever the horizon length yields an infeasible solution to the subproblem, it may be increased at that iteration and the Dynamic Programming routine may then be repeated).

A time limit of 5 minutes was set for each problem run with either heuristic. For Heuristic 1, a maximum of 20 iterations of the subgradient ascent were allowed. The largest size problem that Heuristic 1 could solve in this time limit was J = 15, A = 2, K = 3, M = 5. It took 102.092 seconds of CPU time. Heuristic 2 can solve problems with J = 20, A = 4, K = 6, M = 6 within the 5 minutes time limit. However, due to the computational expense involved, larger problems were not attempted.

The two heuristics were tested using a set of 48 problems to determine the effects of varying J, A, K and M. The different values used were J = 3,6,9,12,15 , A = 2,4,6 , K = 3,6 , M = 3,5,6,8. Using these values, 48 problems were generated. The results are summarized in Tables 7.2 and 7.3.

The run times for the specialized initial job-ARC selection and scheduling routines are dependent on the problem size with the total number of jobs (J) having the most noticeable effect. The initial job-ARC selection heuristic required, on an average, 8.18 x 10^{-3} to 0.024 seconds. The results show that a change in any one of the factors J, A, K or M produces a small change in the run time with the number of jobs having the more pronounced effect. This is to be expected, since the number of jobs plays a major role in the main step of the procedure, whereas, the number of ARCs and the number of operations are required only in the initial stages of the algorithm.

The scheduling heuristic, on an average, required 1.55 x 10^{-3} to 2.669 x 10^{-3} seconds for 3 job problems and 24.35 x 10^{-3} seconds for the 15 job problems. The effect of the (number of jobs) x (number of operations) is pronounced, since each operation is considered individually for loading on a machine. The number of ARCs plays no role in the scheduling heuristic and hence has no effect on the run time of the heuristic. An increase in the number of machines can work towards marginally increasing or

decreasing the run time, depending on whether the effect of the increased number of machines or a decreased number of operations per machine dominates. The change is typically less than 0.5 %.

In Heuristic 1, these times are insignificant compared to the time that the Dynamic Programming (DP) procedure requires in the Lagrangean subproblem solution. The difference is an order of magnitude and is to be expected since the DP processes each ARC of all the jobs. Moreover, DP based algorithms are inherently time consuming because of their state space search requirement. In Heuristic 2, the scheduling heuristic run time is a significant factor, since a schedule is drawn up for every job-ARC combination which needs to be evaluated. However, if the initial job-ARC routine draws up a good schedule with a very tight upper bound, then the effect of the scheduling heuristic run time is minimal. In this case, the lower bound techniques outlined in Section 4.2 are very effective in ARC evaluation and actual schedules are never drawn out. Hence, it is possible for the overall run time for Heuristic 2 to drop even though the scheduling heuristic run time increases - this effect can be seen in problems 19 and 20 in comparison to problems 17 and 18. In this case the initial job-ARC heuristic selects the best set of job-ARCs.

The initial job-ARC solution is used by both Heuristic 1 and Heuristic 2. In Heuristic 1, it is used as a starting feasible solution and as an upper bound for the subgradient algorithm. In Heuristic 2, it is used as an upper bound on the makespan. The initial job-ARC heuristic picks the best solution in a large number of problems. In the problems where it does not pick the best solution, the solution is within 20 % of the best solution. This is very helpful for Heuristic 2, since then, the trivial lower bounds fathom several of the ARCs to be evaluated. This can be seen in the low overall run times for

problems 19, 20, 21 and 22 although the scheduling heuristic run times for these problems are higher than for problems 17 and 18, for example.

Heuristic 1 is sensitive to a number of parameters of the subgradient algorithm, the problem data, problem structure and the upper bound. Hence, although a tighter upper bound helps the subgradient algorithm, the effect of the other parameters dominates the lower bound obtained after 20 iterations and the job-ARCs selected. (Each problem was allowed 20 iterations of the subgradient algorithm.) In 2 cases (problems 17 and 18) the lower bound climbed upto the upper bound, confirming the optimality of that solution. In all the other problems the lower bound would have required more iterations to get closer to the upper bound. However, in most of these cases, the best job-ARC set was selected within the 20 iterations. In the few cases where the best set was not obtained (problems 1, 12, 13, 30, 36 and 37), the result was within 10 % of the best solution.

In conclusion, both Heuristic 1 and Heuristic 2 are able to solve much larger problems than is possible using MIP packages. While Heuristic 1 uses a Lagrangean relaxation and subgradient based approach for job-ARC selection, Heuristic 2 uses a partial enumeration approach for this purpose. The final schedule in both methods is a heuristic schedule, and hence, there is no guarantee of an optimal solution. For the smaller problems the best solutions were checked graphically and appeared to be optimal. In terms of both the solution quality and the run time, Heuristic 2 performs better than Heuristic 1. A number of problems [31, 32, 39, 40 - 44, 45, 47, 48] could not be solved by Heuristic 1 in the 5 minute time limit whereas Heuristic 2 solved them in a few seconds. It is possible to further improve the run time of Heuristic 2 using stronger bounds and a less stringent fathoming criterion. This will permit the solution of much larger problems. In Heuristic 1 , the Dynamic Programming subproblem solution is the

time consuming section and if an equally effective but less time consuming solution method can be implemented, Heuristic 1 would be capable of handling much larger problems in a reasonable amount of time.

Table 7.1. Results of runs with the MPSX package.

# of jobs	# of ARCs	# of opns. per ARC	length of horiz.	LP soln. value	optimal makespan	CPU time (mins.)
3	2	3	10	0.4252	7	1.43
4	3	3	7	-	unsolved	> 5
2	2	3	6	0.4082	5	0.03
4	2	3	9	0.7236	9	0.41
4	3	3	6	0.4620	5	1.61
3	2	3	9	1.871	9	0.17
3	2	3	9	0.7736	9	0.40
3	2	3	5	0.2276	4	0.20

Table 7.2. Initial job-ARC and scheduling run times.

Problem number	J	A	K	M	Initial job - ARCs time (secs) x 0.001	Sched. routine time(secs) x 0.001
1	3	4	3	3	0.818	1.55
2	3	4	3	5	0.889	1.667
3	3	4	6	6	0.915	2.043
4	3	4	6	8	0.752	2.529
5	3	6	3	3	4.812	1.685
6	3	6	3	5	5.052	1.747
7	3	6	6	6	23.5	2.669
8	3	6	6	8	24.1	2.531
9	6	2	3	3	0.873	2.68
10	6	2	3	5	1.069	2.573
11	6	2	6	6	0.896	4.685
12	6	2	6	8	1.007	3.876
13	6	4	3	3	8.94	2.62
14	6	4	3	5	1.099	2.391
15	6	4	6	6	0.943	3.848
16	6	4	6	8	0.916	3.845

Table 7.2. Initial job-ARC and scheduling run times. (..contd.)

Problem number	J	A	K	M	Initial job - ARCs time (secs) x 0.001	Sched. routine time(secs) x 0.001
17	6	6	3	3	1.018	2.57
18	6	6	3	5	1.039	2.415
19	6	6	6	6	0.997	3.1
20	6	6	6	8	0.996	2.974
21	9	2	3	3	1.153	4.665
22	9	2	3	5	1.186	4.663
23	9	2	6	6	1.302	6.361
24	9	2	6	8	1.351	8.371
25	9	4	3	3	1.318	6.361
26	9	4	3	5	1.35	4.837
27	9	4	6	6	1.256	6.380
28	9	4	6	8	1.241	7.951
29	9	6	3	3	1.285	4.724
30	9	6	3	5	1.275	5.601
31	9	6	6	6	1.328	9.199
32	9	6	6	8	1.298	9.194

Table 7.2. Initial job-ARC and scheduling run times. (..contd.)

Problem number	J	A	K	M	Initial job - ARCs time (secs) x 0.001	Sched. routine time(secs) x 0.001
33	12	2	3	3	1.509	14.878
34	12	2	3	5	1.531	14.122
35	12	2	6	6	1.609	21.412
36	12	2	6	8	1.668	17.552
37	12	4	3	3	1.675	15.138
38	12	4	3	5	1.660	14.054
39	12	4	6	6	1.645	28.694
40	12	4	6	8	1.653	24.816
41	12	6	3	3	1.840	11.706
42	12	6	3	5	1.693	15.135
43	12	6	6	6	1.813	26.241
44	12	6	6	8	1.963	30.279
45	15	2	3	3	1.981	23.862
46	15	2	3	5	1.918	32.229
47	15	2	6	6	2.033	41.037
48	15	2	6	8	1.964	39.456

Table 7.3. Results of Heuristic 1 and Heuristic 2.

Problem number	Initial job-ARC makespan	Lagrangean Relxn. bound	Heuristic 1 run time (secs)	Heuristic 1 makespan	Heuristic 2 run time (secs)	Heuristic 2 makespan
1	7	0.0	6.367	7	0.006858	5
2	5	0.0	5.514	5	0.007495	5
3	10	10.0	19.45	10	0.009038	10
4	10	10.0	21.443	10	0.009010	10
5	6	0.5069	11.858	6	0.007929	6
6	7	4.9974	8.887	5	0.008157	5
7	12	2.2595	61.078	12	0.011151	12
8	12	0.0	64.758	12	0.011419	12
9	6	0.0	8.2265	6	0.028511	6
10	4	0.0	9.0934	4	0.030900	4
11	10	3.7358	40.682	10	0.085430	10
12	11	0.0	65.549	11	0.095421	10
13	6	0.0	8.097	6	0.029317	5
14	4	0.0	19.329	4	0.036468	4
15	10	0.0	338.32	10	0.067945	10
16	14	0.0	123.187	14	0.034047	14
17	9	0.222	41.04	7	0.162200	7
18	7	0.2857	28.5	6	0.120200	5

Table 7.3. Results of Heuristic 1 and Heuristic 2. (..contd)

Problem number	Initial job-ARC makespan	Lagrangean Relxn. bound	Heuristic 1 run time (secs)	Heuristic 1 makespan	Heuristic 2 run time (secs)	Heuristic 2 makespan
19	8	0.315	43.5	8	0.072300	8
20	8	0.315	44.367	8	0.080600	8
21	12	0.269	4.665	12	0.047200	12
22	8	0.403	4.663	8	0.043600	8
23	14	0.421	6.361	13	0.222100	13
24	12	0.491	8.371	11	0.564500	11
25	11	0.1818	5.548	9	0.168100	9
26	6	0.333	4.837	5	0.163200	5
27	8	0.1458	6.38	8	0.109900	8
28	7	0.1667	7.951	7	0.138000	7
29	9	0.3827	92.697	9	0.252900	9
30	8	0.4306	38.694	8	0.711600	7
31	9	-	> 5 mins.	-	1.076000	9
32	8	-	> 5 mins.	-	1.156000	8
33	16	0.1979	73.0064	16	0.118000	16
34	10	0.3167	35.6306	10	0.083400	10
35	21	0.4323	271.3816	21	2.042600	21
36	16	0.3294	160.0264	15	0.054010	14

Table 7.3. Results of Heuristic 1 and Heuristic 2. (..contd)

Problem number	Initial job-ARC makespan	Lagrangean Relxn. bound	Heuristic 1 run time (secs)	Heuristic 1 makespan	Heuristic 2 run time (secs)	Heuristic 2 makespan
37	17	0.1292	178.4433	15	1.714500	14
38	9	0.2870	40.9838	8	1.287800	8
39	16	-	> 5 mins	-	6.314100	15
40	16	-	> 5 mins	-	2.187330	14
41	15	-	> 5 mins	-	6.059200	14
42	9	-	> 5 mins	-	5.683600	9
43	11	-	> 5 mins	-	13.90890	10
44	11	-	> 5 mins	-	35.77790	9
45	23	-	> 5 mins	-	0.053970	23
46	17	0.2235	102.092	15	1.148900	15
47	25	-	> 5 mins	-	1.725500	23
48	21	-	> 5 mins	-	2.099600	20

8. Conclusions.

In this effort, we have considered the problem of parts mix selection and sequencing in a Flexible Manufacturing System. Two solution approaches were investigated. The first method, which we call Heuristic 1, uses a Mixed Integer Programming model. The solution of this model uses a combination of Integer Programming techniques and heuristic methods. The second approach, which we call Heuristic 2, uses a quicker enumeration type of method to select a good job-ARC mix. Both heuristics employ a specialized job-shop heuristic to draw up a final schedule. The objective in our study is the minimization of makespan.

The two heuristics were tested using a set of 48 randomly generated problems. These problems were generated to bring out the effect of varying the number of jobs, number of ARCs per job, the number of operations per ARC and the total number of machines on the two heuristics. In terms of solution quality, both Heuristic 1 and Heuristic 2 performed well. Heuristic 1 is sensitive to a number of parameters of the subgradient algorithm, the problem data, the problem structure and the upper bound used. Hence, each one of these factors affects the solution quality and the run time of this heuristic.

The Dynamic Pogramming procedure used in Heuristic 1 is a computationally intensive procedure and controls the performance of this heuristic both in terms of increasing the run time and improving the solution quality. With this procedure removed from the subproblem solution in Heuristic 1, the solution quality is poor although the run time improves considerably. The run time of Heuristic 2 is affected by the problem size. This being an enumeration type of procedure, its solution quality is unaffected by the problem size. The scheduling heuristic and the efficiency of the enumeration technique used to generate the feasible job-ARC combinations control the performance of this heuristic. The largest problem solved in a time limit of 5 minutes by Heuristic 1 had 15 jobs, a maximum of 2 ARCs per job, 3 operations per ARC and 5 machines. In this same time limit Heuristic 2 could solve a problem with 20 jobs, a maximum of 4 ARCs per job, 6 operations per ARC and 6 machines.

Improvements, via further research, can be made in Heuristic 1 by two different approaches, both aimed at reducing or completely eliminating the computational expense of the Dynamic Programming procedure. One approach could be to modify the relaxation structure of Heuristic 1 and solve a different relaxation so as to avoid the subproblem requiring a Dynamic Programming solution. Alternately, one could look for a more efficient procedure which results in a good subproblem solution for the given relaxation structure. Improvements to Heuristic 2 can be directed towards the development of a more efficient enumeration procedure through the use of stronger bounds and possibly a faster scheduling heuristic.

9. References.

Akella, R., Choong, Y. and Gershwin, S. B., "Performance of Hierarchical production Scheduling Policy," **Proc. of the first ORSA/TIMS Conference** , Ann Arbor, Michigan, (1984).

Ammons, J. C., Lofgren, C. B.,and McGinnis, L. F., "A Large Scale Work Station Loading Problem," **Proc. of the first ORSA/TIMS Conference** , Ann Arbor, Michigan, (1984).

Anderberg, M. R., **Cluster Analysis for Applications** , Academic Press, New York, 1973.

Barr, A. and Feigenbaum, E. A., (Editors), **The Handbook of Artificial Intelligence** , Vol. 1-3, HeurisTech Press, Stanford, California, (1983).

Bazaraa, M. S. and Sherali, H. D., "On the Choice of Step Size in Subgradient Optimization," **EJOR** ,7, 380 - 388, (1981).

Barash, M. M., "Computerized Manufacturing Systems for Discrete Products," **The Handbook of Industrial Engineering** , Edited by Gavriel Salvendy, John Wiley and Sons, (1982).

Baskett, F., Chandy, K. M., Muntz, R. R., and Palacios, F. G., "Open Closed and Mixed Network of Queues with Different Classes of Customers," **JACM** , 22, 2, 248-260, (April, 1975).

Berrada, M. and Stecke, K. E., "A Branch and Bound Approach for FMS Machine Loading," **Proc. of the first ORSA/TIMS Conference** , Ann Arbor, Michigan, (1984).

Bullers, W. I., Nof, S. Y. and Whinston, A. B., "Artificial Intelligence in Manufacturing and Control," **AIIE Transactions** , 12, 4, 351 - 363, (1980).

Buzacott, J. A. and Shanthikumar, J. G., "Models for Understanding Flexible Manufacturing Systems," **AIIE Transactions** , 12, 4, 339-350, (December, 1980).

Buzacott, J. A. and Yao, D. W., "Flexible Manufacturing Systems : A Review of Analytical Models," **Management Science** , 32, 7, 890 - 905, (1986).

Camerini, P. M., Fratta, L. and Maffioli, F., "On Improving Relaxation Methods by Modified Gradient Techniques," **Mathematical Programming Study 3** , 26 - 34, (1975).

Chakravarty, A. K. and Shtub, A., "Selecting Parts and Laoding Flexible Manufacturing Systems," **Proc. of the first ORSA/TIMS Conference** , Ann Arbor, Michigan, (1984).

Chang, Y., "Experimental Investigation of Quasi - Realtime Scheduling In Flexible Manufacturing Systems," **Proc. of the first ORSA/TIMS Conference** , Ann Arbor, Michigan, (1984).

Cooper, R. and Jaikumar, R., "Management Control of the Flexible Manufacturing System," **Proc. of the first ORSA/TIMS Conference** , Ann Arbor, Michigan, (1984).

Dubois, D. and Stecke, K. E., "Using Petri Nets to represent production processes," **Proc. IEEE Conf. Decision and Control** , 1062-1067, (Dec. 1983).

Dupont_Gatelmand, C., "A Survey of Flexible Manufacturing Systems," **Journal of Manufacturing Systems, SME** , 1, 1, 1-15, (1982).

El Maraghy, H., "Simulation and Graphical Animation of Advanced Manufacturing Systems," **J. of Mfg. Systems, SME** , 1, 1, 53 - 63, (1982).

Everitt, B., **Cluster Analysis** , Halstad Press, New York, 1980.

Flexible Manufacturing System Handbook - Charles Stark Draper Laboratory, Cambridge, MA, (February, 1983).

Gere, W. S., "Heuristics in Job Shop Scheduling," **Management Science** , 13, 3, 167 - 190, (November, 1966).

Gershwin, S. A., Akella, R. and Choong, Y., "Short Term Scheduling of an Automatic Manufacturing Facility," MIT - LIDS Report LIDS-FR-1356.

Hildebrant, R. R. and Suri, R., "Mathodology and Multilevel Algorithm Structure for Scheduling and Real Time Control of Flexible Manufacturing Systems," **Proceedings 3rd International Symposium on Large Engineering Systems** , Memorial University of Newfoundland, Canada, (July, 1980).

Ho, Y. C. and Cao, X, "Perturbation Analysis and Optimization of Queueing Networks," **The Journal of Optimization Theory and Applications** , 40, 4, 559 - 582, (1983).

Ho, Y. C., Eyler, M. A. and Chien, T. T., "A New Approach to Determine Parameter Sensitivities on Transfer Lines," **Management Science** , 29, 6, 700 - 714, (June , 1983).

Ho, Y. C., Eyler, M. A., Chien, T. T., "A Gradient Technique for General Buffer Storage Design in Serial Production Line," **Int. J. Prod. Res.** , 17, 6, (1979), pp. 557 - 580.

Hutchinson, G. K. and Hughes, J. J., "A Generalized Model of Flexible Manufacturing Systems," Kearney and Trecker Corp., (1977).

Kalkunte, M. V., Kusiak, A., Sarin, S. C. and Wilhelm, W. E., "Flexible Manufacturing Systems : A Review of Modelling Approaches for Design, Justification and Operation," **FMS Methods and Studies : Studies in Management Science and Systems Series** , ed. Kusiak, A., North Holland Publications, (1986).

Kinemia, J. and Gershwin, S. B., "Network Flow Optimization in Flexible Manufacturing Systems," **Proceedings of the IEEE Conference on Decision and Control** , (1979).

Kinemia, J. and Gershwin, S. B., "An Algorithm for the Computer Control of Production in Flexible Manufacturing Systems," **IIE Transactions** 15, 4, 353 - 362, (1983).

Kusiak, A., "Loading Models in Flexible Manufacturing Systems," **Recent Developements in Flexible Manufacturing Systems and Allied Areas** , North Holland, Amsterdam, (1984).

Kusiak, A., "The Part Families Problem in Flexible Manufacturing Systems," **Proc. of the first ORSA/TIMS Conference** , Ann Arbor, Michigan, (1984).

Lin, C. S. and Lu, C. Y., "The Scheduling Problem in Random Flexible Manufacturing Systems," **Proceedings of the First ORSA/TIMS Special Interest Conference** , Ann Arbor, Michigan, (1984).

Mayer, R. J. and Talavage, J. J., "Simulation of a Computerized Manufacturing System," Report No. 4, NSF Grant No. APR74-15256, (November, 1976).

Morton, T. E. and Smunt, T. L., "A Planning and Scheduling System for Flexible Manufacturing," **FMS : Methods and Studies** , Ed. Kusiak, A., North Holland, 151 - 164, (1985).

Moore, J. M. and Wilson, R. C., "A Review of Simulation Research in Job Shop Scheduling," **Production Inventory Management,** 8, 1 - 10, (1967).

Narahari, Y. and Vishwanathan, N., "Analysis and Synthesis of Flexible Manufacturing Systems Using Petri Nets," **Proc. of the first ORSA/TIMS Special Interest Conference,** An Arbor, Michigan, (Aug. 1984).

Nof, S. Y., Barash, M. M. and Solberg, J. J., "Operational Control of Item Flow in Versatile Manufacturing Systems," **Int. J. of Production Research** , 17, 5, (1979).

Nof, S. Y., Whinston, A. B. and Bullers, W. I., "Control and Decision Support in Automatic Manufacturing Systems," **IIE Transactions** , Vol. 12, 2, 156-167, (1980).

Panwalker, S. S. and Iskander, W., "A Survey of Scheduling Rules," **Operations Research** , 25, 1, 45 - 61, (January, 1977).

Peterson, J. L., "Petri Net Theory and the Modelling of Systems," Prentice Hall Inc., Englewood Cliffs, NJ, (1981).

Phillips, D. T. and Heisterbey, R. J., "Definition, Developement and Implementation of a Generalized Manufacturing Simulator," Report No. GEMS-3-77, Texas A&M University, College Station, TX, (1977).

Sarin, S. C. and Dar-El, E. M., "Approaches to the Scheduling Problem in FMS," **IIE Proceedings** , Atlanta, 1984.

Solberg, J. J., "A Mathematical Model of Computerized Manufacturing Systems," presented at the **4th International Conference on Production Research** , Tokyo, (August, 1977).

Stecke, K. E., "Formulation and Solution of a Nonlinear Integer Production Planning Problem for Flexible Manufacturing Systems," **Management Science** , Vol. 29, 3, 273 - 278, (March 1983).

Stecke, K. E. and Solberg, J. J., "Loading and Control Policies for a Flexible Manufacturing System," **Int. J. of Production Research** , 19, 5, 481 - 490, (1981).

Suri, R., "Robustness of Queueing Network Formulae," **J. ACM** , 30, 3, 564 - 594, (July 1983).

Suri, R. and Xao, X., "Optimization of Flexible Manufacturing Systems using New Techniques in Discrete Event Systems," **Proc. 20th Allerton Conference on Communication, Control and Computing** , Monticello, IL, (October, 1982).

Vaithianathan, R., "Scheduling in Flexible Manufacturing Systems," **Proc. 1982 Fall Ind. Engg. Conf.** , Cincinnati, OH, (November, 1982).

Wilhelm, W. E. and Sarin, S. C., "Models for the Design of Flexible Manufacturing Systems," **Proceedings of the Annual IIE Conference,** , Louisville, Kentucky, (1983).

Yao, D. D., "An FMS Network Model with State- Dependent Routing," **Proc. of First ORSA/TIMS Conference on Flexible Manufacturing Systems** , Ann Arbor, MI, (1984).

Appendix A. Objective functions for the MIP.

The different objective functions which can be used with the Mathematical Model are given below :

Minimize the total idle time for the jobs (or the total flowtime).

$$\text{Minimize} \quad \sum_{j=1}^{N} \max_{t} \{ t \sum_{a \in A_j} x_{jan_j^a t} \} \quad - \quad \sum_{j=1}^{N} \sum_{a \in A_j} \sum_{k=1}^{n_j^a} P_{jak} z_{ja}$$

Minimize the total machine idle time (makespan times the number of machines - total processing time).

$$\text{Minimize} \quad \left\{ \text{M maximum}_{(j,t)} \{ t \sum_{a \in A_j} x_{jan_j^a t} \} \quad - \quad \sum_{j=1}^{N} \sum_{a \in A_j} \sum_{k=1}^{n_j^a} P_{jak} z_{ja} \right\}$$

Minimize the total machine idle time over a fixed shift duration i.e. minimize (number of machines times shift duration - total processing time) = maximize total processing time.

$$\text{Maximize} \quad \sum_{j=1}^{N} \sum_{a \in A_j} \sum_{k=1}^{n_j^a} P_{jak} z_{ja}$$

Minimize the maximum machine idle time or maximize the minimum machine utilization.

$$\text{Minimize} \quad \left\{ \max_{(j,t)} t \sum_{a \in A_j} x_{jan_j^a t} \; - \; \min_{m} \sum_{j=1}^{N} \sum_{a \in A_j} \sum_{k \in S_m^{ja}} P_{jak} z_{ja} \right\}$$

where

S_m : {(j,a,k) : operation k of ARC $a \in A_j$

of job j requires machine m } for $m = 1, \ldots, M$

S_m^{ja} : $\{k:(j,a,k) \in S_m\}$ $j = 1, \ldots, N, a \in A_j, m = 1, \ldots, M$

The vita has been removed from the scanned document