AN EVALUATION OF FLOW SHOP SCHEDULING HEURISTICS

By

FRANCIS DEAN BOOTH

Bachelor of Science in Business Administration
Simpson College
Indianola, Iowa
1958

Master of Business Administration
Arkansas State University
State University, Arkansas
1983

Submitted to the Faculty of the Graduate College
of the Oklahoma State University
in partial fulfillment of the requirements
for the Degree of
DOCTOR OF PHILOSOPHY
December, 1987

AN EVALUATION OF FLOW SHOP

SCHEDULING HEURISTICS

Thesis Approved:

Scott Turner
Thesis Advisor

Ramesh Sharda

Kenneth Shiang

Wm Wards

James T. Jackson

Norman N. Durham
Dean of the Graduate College

PREFACE

This study evaluates flow shop scheduling heuristics in two phases. Phase one compares the individual performance, over a set of 160 randomly generated problems, of six heuristics taken from current literature.

Phase two uses the six heuristics plus an ordinal sequence to initiate a neighborhood search for a better solution. Six neighborhood generation schemes and two improvement rules are tested over the same problem set used in phase one.

Significant differences were found due to individual heuristic performance, number of jobs to be scheduled, number of machines to be utilized, combinations of initializing heuristics and neighborhood generating schemes, and improvement rules. The results may have practical applicability in the scheduling of jobs through the manufacturing cells of organizations employing group technology.

I wish to express my sincere gratitude to all who made my stay at Oklahoma State University such a productive and rewarding experience. I am particularly indebted to my major advisor, Dr. J. Scott Turner, for leading me to an intense interest in the topic under study, and for his sage advice, encouragement, and invaluable assistance during the course of this study.

I am also grateful to my other committee members, Dr. Mitchell O.

Locks, who chaired the committee until his retirement, Dr. Hon-Shiang Lau, Dr. Ramesh Sharda, Dr. James F. Jackson, and Dr. William D. Warde, for their advice and encouragement during the planning and conduct of this work.

To my wife, Ruth, and children, Tamara, LeAnn, and Timothy, I extend my deepest appreciation for their thoughtful understanding, helpful encouragement, and constant support during this undertaking.

TABLE OF CONTENTS

LIST OF TABLES

CHAPTER I

INTRODUCTION

1.1 General Statement of the Problem

The American Production and Inventory Control Society (APICS) [2] defines a flow shop as follows:

> A shop in which machines and operators handle a standard, usually uninterupted material flow. The operators tend to perform the same operations for each production run. A flow shop is often referred to as a mass production shop, or is said to have a continuous manufacturing layout. The shop layout (arrangement of machines, benches, assembly lines, etc.) is designed to facilitate a good product "flow". The process industries (chemicals, oil, paint, etc.) are extreme examples of flow shops. Each product, though variable in material specifications, uses the same flow pattern through the shop. Production is set at a given rate, and the products are generally manufactured in bulk. (p. 12)

Flow shops can have a variety of processing patterns. Graves, Meal, et al [36] describe a reentrant flow shop as one where products may be routed to a machine or operation more than once in a processing sequence. The most common flow shop problem found in the literature is referred to as the n-job, m-machine (or n x m) flow shop. In this model, the only requirement is that each job be processed by each of the m machines in a given machine sequence. Some jobs may have zero processing time on one or more machines in the given sequence.

Scheduling in a flow shop requires determining the sequence in which available jobs will be processed. There are a number of criteria that can be used to evaluate flow shop schedules. These are

discussed in detail in chapter 2.

It is theoretically possible to enumerate all n! possible sequences by which n jobs might be processed, calculate the objective function for each sequence, and select the sequence which optimizes the objective function. This straight-forward approach works well for very small problems but rapidly grows beyond the bounds of practicality for even today's high speed computers as the number of jobs increases. Complete enumeration of a problem involving only ten jobs requires calculating the objective function value for 3,628,800 different sequences. As a result, solution methods have been sought which offer the potential to reduce the number of sequences to be considered.

Three primary approaches to the solution of flow shop problems have been developed in the literature. The first two of these, implicit enumeration and integer linear programming, are capable of determining optimal solutions. These methods still require an inordinate amount of computational effort and, for problems of realistic size, too much computer processing time to be of much practical use. This has led to the development of heuristic methods with which this study is concerned. Heuristic methods determine a good (near optimum) but not necessarily optimal solution.

The general purpose of this study is to evaluate a number of heuristics in much greater depth than is currently found in the literature. Heuristics can be generally classified into one or a combination of two classes. Some find a single sequence which is as near optimum as possible. Others attempt to improve an initial solution by searching one or more neighborhoods of related sequences.

A few combine these two approaches into a single multiple stage procedure. This study will analyze a number of starting procedures, several methods of forming neighborhoods for subsequent search, and the interaction between starting and search procedures. In addition, it will analyze the tradeoff between solution efficiency with respect to the optimal solution or best heuristic solution and computational effort as reflected by the computer processing time.

1.2 Relevant Studies

The literature dealing with job scheduling, in general, and with flow shop scheduling, in particular, is extensive. A discussion of existing literature relevant to this study is given in chapter 2.

1.3 Justification for this Study

Much research effort has been devoted to the flow shop problem over the past three decades. The problem has had great interest from a theoretical standpoint because many of the factors which affect the "pure" flow shop model are common to other scheduling models that have had more practical applicability. With the exception of the process industries, there were very few instances of a "pure" flow shop to be found. Thus, the primary benefit to be derived from flow shop research was the insight and understanding gained which could then be transferred to other scheduling problems of a more practical nature. There now appears, however, to be rapidly developing an industrial methodology for which flow shop scheduling is particularly appropriate.

The United States and other industrial nations exhibit an

increasing trend toward widespread adoption of group technology as a means of increasing the productivity of certain manufacturing processes. APICS [2] defines group technology as follows:

> An engineering and manufacturing philosophy which identifies the "sameness" of parts, equipment, or processes. It provides for rapid retrieval of existing designs and anticipates a cellular type production equipment layout. (p. 13)

The basic concept of group technology is to identify a family of parts or products which, because of their inherent similarity, require essentially the same production processes. Similarity, as used here, refers to a wide variety of product or process characteristics ranging from similarity of the end product to simply similarity of the operations required to produce widely different products. The machines necessary to accomplish these processes are segregated into a production cell and arranged in a way that facilitates the production of a particular family of products. This is one method of implementing Skinner's [67] "plant within a plant" concept of the focused factory. Group technology and manufacturing cells can significantly reduce the setups required in that they are very similar for each member of the product group. Material handling costs often show a marked reduction because parts need not be moved around the plant from one process to another. This also avoids a lengthy queue at each successive process, a fact which greatly reduces work-in-process inventory levels and product lead time.

Group technology appears to be the epitome of a "pure" flow shop. The different parts or products which are produced in any one manufacturing cell are inherently similar in their manufacturing characteristics. All items produced require essentially the same

sequence of machines. Admittedly, group technology is not appropriate for all production processes. Its application is limited primarily to repetitive manufacturing industries. In those cases, however, where it is appropriate, group technology appears to offer an opportunity for practical application of the results of flow shop scheduling research. Thus, additional efforts in this area can be of practical as well as theoretical value.

Although existing literature contains much discussion comparing one heuristic with another, there is no systematic analysis of heuristic methods. Most authors compare their proposed heuristic with the best performing previously existing method. They make little attempt to compare the solutions achieved to the optimal or best solution nor do they attempt to analyze the tradeoff between the amount of improvement achieved with respect to other heuristics or the best solution and the computational effort required to obtain the solution. It is this gap in the literature that this study is intended to fill.

1.4 Specific Objectives of this Study

The specific objectives of this study are to answer the following questions:

a. With respect to initialization procedures -

(1) Which initialization procedure is best as a stand alone procedure and from what standpoint is it better?

(2) Does the choice of initialization procedure depend upon the search procedure to be subsequently employed?

b. With respect to neighborhood search procedures -

(1) Does the neighborhood size account for the effectiveness of the search procedure?

(2) Are there diminishing returns for larger neighborhoods?

(3) What tradeoffs, in terms of computational speed versus solution effectiveness, are involved in using a first improvement rule rather than a best improvement rule?

1.5 Scope and Limitations

This study is limited to permutation schedules. Previous research by Baker [6] and others has indicated that requiring the same processing sequence on all machines has little impact on the value of the objective function being optimized. Optimal in this study will, therefore, refer to the best permutation schedule.

The optimization criteria to be used is that of minimizing makespan. See section 2.4 for further discussion of this and other optimization criteria.

This study will be limited to simulated problems of selected sizes with processing times to be generated from appropriate distributions. Problem sizes will be selected with sufficient range to permit at least limited generalization of the findings. Distributions of processing times will reflect those most frequently found in current literature. A subsequent study will employ a distribution of processing times which reflect the situation most likely to exist in group technology applications.

To permit consistent comparison of simulation results, an optimal permutation schedule for each problem in the problem set will be sought, together with the associated makespan, through an integer

linear programming model. All heuristc solutions will be determined on an IBM 3081K using programs written in Fortran. These programs, together with the routine which randomly generates the problem set, are given in the appendix.

CHAPTER II

LITERATURE REVIEW

2.1 Introduction

Job scheduling has been subject to much intensive research study for more than thirty years. There is a wealth of literature dealing with the topic, much of which specifically treats the flow shop scheduling problem that is the object of this study. Numerous books and professional journal articles discuss and review the existing literature. Some examples are books by Baker [6], Conway et al. [20], French [28], and Rinnooy Kan [60], and articles by Bakshi and Arora [9], Graves [35], and Sisson [66]. The discussion which follows is not intended to be an exhaustive review of the literature but will, instead, cite typical examples from the voluminous literature in this field.

2.2 Problem Assumptions

It is the assumptions with respect to problem characteristics and parameters that distinguishes the flow shop problem from other scheduling problems. Typical assumptions can be found in Conway et al. [20], Dudek and Teuton [25], French [28], and Sisson [66] among others. The following list is that given by Sisson (pp.298-299):

1. No machine may process more than one operation at a time.

2. Each operation, once started, must be performed to completion.

3. A commodity is an entity; that is, even though the commodity represents a lot of individual parts, no lot may be processed by more than one machine at a time.

4. A known finite time is required to perform each operation and each operation must be completed before any operation which it must precede can begin.

5. The time intervals for processing are independent of the order in which the operations are performed.

6. Each commodity must be processed by a designated sequence of machines, this sequence also being called 'the technological ordering' or 'the routing'.

7. There is only one of each type of machine.

8. A commodity is processed as soon as possible subject only to routing requirements given above.

9. All jobs are known and are ready to start processing before the period under consideration begins.

10. The time to transport commodities between machines is negligible.

11. In-process inventory must be allowable.

Dudek and Teuton [25] include one additional assumption. They assume a common job processing order on all machines. This assumption creates what is called a permutation flow shop which reduces the number of possible sequences from $(n!)^m$ to $(n!)$. Baker [6] and others point out that permutation schedules do not guarantee an optimal solution but are capable of providing good solutions that are very nearly optimal. Conway et al. [20], citing work by Heller [39], provide additional justification for considering only permutation schedules.

Assumption 8 above produces what many researchers refer to as

active schedules. Active schedules are those in which all jobs are started on each machine as soon as both the job and the machine are available. The dominance of active schedules was first noted by Giffler and Thompson [31]. Conway et al. [20] also refer to this situation as a non-delay schedule. Such schedules are appropriate when processing technology permits delays between operations and when in-process inventory is allowable (assumption 11). French [28] provides an excellent discussion of active, semi-active, and non-delay schedules.

Sisson [66] makes the following observation with respect to the realism of assumptions:

> It might be emphasized that in most real situations some of the assumptions...do not apply and, in many, none do. Nevertheless, there is good indication that the model resulting from adopting these assumptions characterizes the heart of the sequencing problem. (p. 304)

2.3 Problem Complexity

A problem that can be solved in a determinable number of steps, the number of which is no more than polynomially related to problem size, is said to be P-complete. Other problems fall into a class, known as NP-complete, which cannot be solved in a polynomially bounded number of steps. Garey and Johnson [30], among others, show that the flow shop sequencing problem is NP-complete. This finding is important to flow shop research in that it indicates that there is no solution to the problem that can be achieved in a polynomial number of steps. It is this realization that has led to much of the effort to develop efficient heuristics that are likely to produce good but not necessarily optimal solutions.

2.4 Optimization Criteria

Researchers over the years have used a variety of optimization criteria for flow shop problems. Conway et al. [20] cite the average or maximum of completion time or flow time as possible criteria for flow shops. With the assumption that all jobs are available for processing at time zero (assumption 9), flow time and completion time are equivalent. Szwarc [70] and Panwalkar and Kahn [56] use mean completion time criteria. Bansal [11] uses the sum of completion times which is equivalent to using mean completion time. This criteria is used in situations, such as repair shops, where returning each item to service in the shortest possible time is critical.

Several authors specifically recognize that the optimization criteria ultimately involves some function of costs. Gupta and Dudek [38] propose an opportunity cost criteria which is a combination of processing cost, machine idle cost, and a penalty cost for late jobs. Sisson [66] notes that the ultimate desire is to optimize an objective of the organization, profits for example, but that this requires a detailed knowledge of how the specific situation relates to the overall objective. The relationship is unique to each organization and is very difficult to estimate with any accuracy. In research, one usually chooses to optimize a lesser criteria chosen in some reasonable way. Following this logic, most researchers have chosen to minimize completion time, also known as schedule time or makespan. Makespan is easy to apply and has a stable relationship to other criteria such as machine idle time, machine utilization rates, and in-process inventory costs. Rinnooy Kan [60] shows that minimizing

makespan is equivalent to maximizing the amount of work in progress at a given time, minimizing the total idle time or the weighted sum of idle times, and maximizing the mean utilization of machines. Following a lengthy discussion of scheduling costs, he concludes that the use of makespan can be reasonably justified on economic grounds. French [28] provides a similar comparison of optimization criteria.

In a survey of industry conducted in 1971, Panwalkar, Dudek, and Smith [55] found minimizing makespan to be the second most popular criteria among respondents, second only to the criteria of meeting due dates.

2.5 Flow Shop Problem Solution Approaches

Over the past thirty years, a number of approaches to solution of the flow shop problem have been developed. Heller [39], as quoted by Sisson [66], summarized the objective of each of these approaches as follows:

> The objective of many previous investigations...is to find an arrangement that minimizes the processing time...as a function of given job times... This objective is not the whole story. We must ask the question: Can we find an order relation that minimizes the processing time such that the number of arithmetic and logical operations to obtain this minimum order is very much smaller than the number of arithmetic and logical operations needed to enumerate all processing times and their corresponding order relations? (pp. 305-306)

Sequencing research is thus concerned not only with developing algorithms which produce optimal or near optimal solutions but with methods that can produce such solutions with computational economy in practical situations. One obvious method of producing an optimal sequence is to enumerate all possible sequences, compute the objective

function value associated with each, and choose the sequence which optimizes the objective function. It is also obvious that, for problems of any size, the n! possible sequences for a permutation flow shop rapidly exceed the bounds of practicality for even a high speed computer. Therefore approaches have been sought which reduce the number of sequences which must be considered.

2.5.1 Johnson's Two-Machine Algorithm

Johnson [341] developed an algorithm which produces an optimal makespan solution to the n x 2 flow shop problem. His procedure involves finding the shortest processing time among all jobs on both machines. The job with which this time is associated is scheduled first in the sequence if the shortest time occurs on the first machine and last in the sequence if it occurs on the second machine. The remaining jobs are then searched for the next shortest processing time, the job is scheduled accordingly, and the process is repeated until all jobs are assigned a sequence position. This simple algorithm can be extended to optimize n x 3 problems under certain restrictive conditions. Johnson extended his algorithm to cases where the second machine was dominated by either the first or the third. He then applied the two machine procedure to artificial times created by adding the processing times for each job on the first two (machines 1 and 2) and last two (machines 2 and 3) machines. Burns and Rooker [17] further extended the three machine problem to cases where the processing times on the second machine were less than the lowest processing time on either the first or third machine. Attempts to

extend Johnson's algorithm to optimizing schedules for more than three machines have met with no success. His three machine procedure does, however, form the basis for some sub-optimizing heuristics as will be discussed later in this chapter.

2.5.2 Implicit Enumeration Methods

Implicit enumeration methods which employ branch and bound techniques have become quite common in management science applications. These methods use various bounding procedures to reduce the number of combinations which must be explored in seeking an optimal solution. Because whole branches of a search tree can be eliminated when they offer no potential to improve an existing solution, these methods are also called elimination methods in some of the literature.

The implicit enumeration approach was first applied to the traveling salesman problem by Little et al. [46]. Brooks and White [15] applied this approach to production scheduling in general but noted that the procedure was too long to provide computationally economic solutions on the then existing computers. Lomnicki [47] applied the procedure to the three-machine flow shop problem and Ignall and Schrage [40] extended the application to problems with more than three machines and noted that it was only practical to consider permutation schedules in such cases. Both Lomnicki and Ignall and Schrage proposed bounds that were machine based. Bounds were calculated from the total processing time remaining on a given machine plus the minimum run-out time for a job from that machine. McMahon and Burton [50] proposed job based bounds which use the total

processing time for a job. They also determined that an optimal solution could be found more quickly if one of the heuristics were used to initially order the jobs to provide a near optimal starting sequence. Balas [10] developed an implicit enumeration algorithm which uses disjunctive graphs as the basis for its bounding procedures.

Elimination methods are reviewed by Szwarc [69]. This approach constructs a set of dominant schedules to eliminate other sequences which cannot contain an optimal solution. Enumeration is required only for the dominant set. Baker [8] describes how these models may perform in moderate sized problems and shows that the size of the dominant set is still too large to provide computational efficiency. In a separate article, Baker [7] finds that a combination of branch and bound with elimination methods gives greater computational efficiency than either method alone. Following Baker's lead, Bestwick and Hastings [12] made some minor changes to previous bounding procedures while combining the two methods. They also noted the appropriateness of the flow shop to group technology. However, their 'real problem' example reflects more instances of zero processing times (item not processed by a given machine) than would seem typical of group technology and cellular manufacturing as we know it today. Working with larger problems, Lagaweg et al. [42] also found that a combination of branch and bound with elimination methods provided greater computational efficiency. They concluded that, with few machines, problems with up to 50 jobs could be solved rather quickly. An increase in the number of machines, however, made lower bounds less reliable and increased solution times drastically. They also

expressed some concern with the tradeoff between the sharpness of the bound and the computation required to produce it. Stronger bounds eliminate more nodes of the search tree. However, if the effort required to compute them becomes excessive, it may be better to search through more nodes using weaker but more quickly computed bounds.

2.5.3 Integer/Mixed Integer Linear Programming

Based on some earlier work by Dantzig [23] and Markowitz and Manne [49], Gomory [33] applied integer linear programming to the scheduling problem. A minor revision to his original algorithm appears in Gomory [34]. This early work dealt with the more general problem of job shop scheduling of which the flow shop is a special case. This work was extended by Bowman [14]. The all integer linear prgramming models for scheduling were initially applied to problems involving three or fewer machines. As formulated by Bowman, the integer (0-1) variables denoted whether a job was being processed by a given machine during a specific increment in time. A maximum number of time periods was chosen between the sum of the processing times on the longest product and the simple sum of all processing times. A 4 x 3 problem would require from 300 to 600 variables depending on the number of time increments selected. The model required constraints to ensure the required processing time for each job on each machine, constraints to ensure that each machine was processing only one job at a time, job sequence constraints, and constraints to guarantee uninterupted processing on a machine. The objective function included penalty weights for the later time increments to ensure that processing was completed as soon as possible (minimum makespan).

Although there was no restriction on problem size, the number of variables required made the computation necessary to solve a problem of even 'modest' size quite large. It was noted without empirical results that solution of the dual problem might reduce computational effort.

Wagner [74] proposed an integer programming model for the job shop which greatly reduced the number of variables required. He maintained the all integer formulation by requiring integer processing times. He introduced the concepts of machine idle time and job delay time which were to be key elements in later models. Wagner specifically applied his model to the flow shop and, in doing so, noted that minimizing the idle time on the last machine would minimize makespan. He also noted a fundamental relationship that must exist between two consecutive jobs and two consecutive machines, a relationship which forms the basis for the principal constraints in the model. Story and Wagner [68] report some computational experience using this model for a three machine flow shop. They also explore the potential of simply rounding the non-integer linear solution. Manne [48] makes some refinements to the job shop integer programming model and notes the potential of this problem for solution by mixed integer programming, algorithms for which were then not yet available. After noting the excessive time required to obtain the optimal solution to the three machine problem using Wagner's model, Giglio and Wagner [32] compare the linear programming results to the results obtained using several other methods that produce near optimal solutions. These near optimal methods will be discussed further in a subsequent section of this chapter.

A further attempt to refine the integer programming model is found in Baker [6]. Baker's model is very similar to the one proposed earlier by Wagner [74]. Unfortunately, Baker's model contains some critical omissions which cause it to produce incorrect results. A similar omission also occurs in a statement and discussion of Wagner's model by Rinnooy Kan [60]. A correct formulation of the mixed integer programming model can be found in French [28].

Still another formulation of the mixed integer programming model can be found in Turner and Booth [72]. Empirical results to date are too few to permit significant conclusions about the computational efficiency of this model.

In a general report on mixed integer programming models for production scheduling, Bruvald and Evans [16] report that the computational effort involved in solving such models is much more sensitive to the number of integer variables than it is to the number of continuous variables included in the model formulation. The basic drawback with mixed integer programming models for the flow shop problem is similar to the drawback found with implicit enumeration methods. While each method is capable of producing an optimal permutation schedule, there is no accurate way of predicting the computational effort that will be required.

2.5.4 Heuristic Methods

The computational effort, time, and computer resources required to obtain an optimal schedule to the permutation flow shop problem by any of the optimizing methods has proven to be so great for any but the very smallest problems that much of the more recent research

effort has been devoted to the development of heuristics which are capable of producing good (near optimal) but not necessarily optimal schedule sequences.

Heuristics can be easily classified as being one or a combination of two types. Most of the heuristics proposed in the literature produce a single solution which is as near optimal as possible. Unless otherwise indicated, the heuristics discussed below fall into this category. A few use a neighborhood search procedure to improve an initial solution chosen in either an arbitrary or systematic way. Still fewer combine both an initial solution procedure and a neighborhood search procedure. These latter two types will be pointed out as they are encountered below.

Page [53] noted that the scheduling problem was very similar to the sorting problem of data processing and applied some of the methods frequently used for sorting to the job sequencing problem. He proposed three heuristics based on these methods. Two of these, merging and pairing, develop a single solution. The third, exchanging, is a neighborhood search procedure. In merging, strings of successive pairs of jobs are ordered based upon makespan for the pair. In subsequent iterations, the number of jobs per string is increased while the number of strings is decreased until a single ordered string is obtained. In pairing, once the initial ordering of pairs is done, the pairing is regarded as permanent and it is only necessary to order the pairs, quartets, etc. as a whole. Page recognized that this method was unlikely to produce as good a sequence as merging but suggested that the considerable reduction in computational effort might be adequate compensation for the poorer

results. Exchanging starts with an arbitrary or random sequence which is improved by the exchange of adjacent pairs. Exchanging is continued until no improvement is obtained in a complete pass through the last sequence obtained. The computational effort for exchanging is on the same order as that required for merging. The optimization objective of all of Page's heuristics was to minimize makespan.

Palmer [54] suggested that jobs placed first in the schedule sequence should have processing times that display an incresing trend from machine to machine through the technological ordering of machines and that jobs near the end of the sequence should have a decreasing trend. This is a generalization to the m machine case of the situation found in Johnson's two machine algotithm. Palmer defined a slope order index as:

$$S_i = - \sum_{j=1}^{m} \{[m-(2j-1)]/2\} \, t_{ij} \qquad (2.1)$$

where t_{ij} is the processing time for job i on machine j. Jobs are sequenced in decreasing order of S_i. The objective function is to minimize makespan.

Petrov [58] also adapted Johnson's algorithm to the m machine case. Machines are divided into two equal halves with the center machine being included in both halves when the number of machines is odd. The two halves of the processing time matrix are summed for each job. The sums are then treated as if they were processing times on the two machines of Johnson's algorithm. The objective function is to minimize makespan. Petrov gives a set of rules to be applied that are much more complex than Johnson's simple algorithm but which result in the same ordered sequence.

Campbell, Dudek, and Smith [18] have also adapted Johnson's two machine algorithm to the m machine problem. They create a set of m-1 artificial two machine sub-problems, order each according to Johnson's algorithm, and then select the sequence from the resulting set of sequences which produces the best makespan. They create the set of artificial sub-problems by summing the processing times on the k ($1 <= k <= m-1$) first and last machines. If ties occur in any of the m-1 sub-problems, say in sub-problem k, the tie is broken by using the order for the tied jobs created in the k-1st sub-problem. If the tie still cannot be broken, proceed in order through sub-problems k-2, k-3,...,1, then to k+1, k+2,...,m-1. In the rare instances where ties cannot be broken, the authors recommend retaining two or more sequences for this sub-problem. Empirical testing against Palmer's heuristic provided consistently better results but at a cost of greater processing time.

Gupta [37] proposes calculation of a function value for the m machine problem in much the same manner as Palmer's slope order index. The function is defined as:

$$f(i) = \min_{1 <= m <= (M-1)} \frac{A}{(t_{im} + t_{i,m+1})} \qquad (2.2)$$

where $A = 1$ if $t_{im} <= t_{i1}$
$= -1$ otherwise
and M is the number of machines

He then arranges the jobs in ascending order of f(i), breaking any ties in favour of the job with the smallest sum of processing times on all machines. The objective function for this heuristic is to

minimize makespan. Empirical tests against Palmer's heuristic provided consistently better results with approximately the same computational effort.

Krone and Steiglitz [44] propose a heuristic which applies a two-stage neighborhood search procedure. They start with a psuedo-random sequence and subject it to a series of neighborhood searches. In stage one, the neighborhood to be searched is created by removing job j and reinserting it in the ith position for $1 \leq i < j \leq n$. They employ a first improvement rule which retains a new sequence when the first improvement is achieved rather than searching the remainder of the neighborhood for the best improvement. When an improved sequence is found, the procedure continues with indices i and j+1, returning to the beginning of the sequence as necessary. In stage two, all pairs are checked to see if an exchange of positions will provide further improvement. In the empirical tests, the results for each stage were compared to an 'empirical optimum' which was the best of twenty runs using different psuedo-random starting sequences. It was noted that the average additional improvement achieved in stage two ranged from 0 to .66 percent depending on problem size. Although the authors used the minimization of mean completion time as their objective function, the procedure could also be applied with an objective to minimize makespan. Only a change in the calculation of the objective function value would be required. Such a change would reduce the computational effort and the amount of computer storage required because only the completion time of the last job would be required.

Bonney and Gundry [13] extended the concepts of Palmer [54] and

Gupta [37] by noting that Palmer's slope order index and, to some extent, Gupta's functional index, were actually an average of a start slope and end slope of the job profile. In problems where no job waiting was allowed (in-process inventory not permitted), the job profile is independent of sequence. By computing both a start slope and end slope for each job, they were able to search for a sequence which provided the best match between the end slope of one job and the start slope of the next. The sequence was started with the job having the largest start slope. The procedure was simplified by applying Johnson's two machine algorithm to the computed start and end slopes. In doing so, largest slope is substituted for smallest processing time in Johnson's algorithm. The objective function is to minimize makespan. The empirical results indicated that their slope matching heuristic outperformed both Palmer's and Gupta's heuristics when either n or m was large.

Dannenbring [22] proposed three heuristics for the flow shop sequencing problem. His rapid access heuristic can be used to obtain a quick starting solution for the other two. This method uses a weighting scheme similar to Palmer's slope order index and the Campbell, Dudek, and Smith methods. A single two-machine sub-problem is created for which processing times are determined from the weighting scheme. The sub-problem is then solved using Johnson's two-machine algorithm. Defining P_{ij} as the processing time for the i^{th} job on the j^{th} machine in the sub-problem ($i = 1,2,\ldots,n$ and $j = 1,2$) and t_{ij} as the processing time for the i^{th} job on the j^{th} machine in the original problem, the P_{ij}'s are calculated as:

$$P_{i1} = \sum_{j=1}^{m} (m-j+1)\ t_{ij}, \quad P_{i2} = \sum_{j=1}^{m} (j)\ t_{ij} \tag{2.3}$$

Using the results of rapid access as a starting sequence, Dannenbring then searches for an improved solution in a neighborhood defined by exchanging adjacent pairs. His 'rapid access with close order search' heuristic employs a single pass through the adjacent pairs neighborhood. 'Rapid access with extensive search' employs multiple passes by creating new neighborhoods from the best sequence found in the previous search. The heuristic terminates when no improvement is found on a search pass. The objective is to minimize makespan. Empirical testing against several existing heuristics showed that rapid access with extensive search outperformed all others tested on average but required much more computer processing time than most other methods tested.

King and Spachis [42] also used the job profile concept to develop a heuristic which incorporates a weighting scheme for machine idle times. Noting that idle times on machines in the latter part of the machine sequence would tend to have greater adverse effect on makespan, the authors devised a simple weighting scheme which uses the machine sequence number as the weighting factor. This heuristic develops a set of n sequences wherein each job occupies the first position in one sequence. The end profile of the first job is used to select, from the remaining jobs, the one that gives the 'least total weighted between jobs delay'. Trial jobs are left shifted (picture a Ghant chart) as far as possible and weighted machine idle time is computed. The job which gives the smallest total weighted machine

idle time is selected to go next in the sequence. The new end profile (after left shifting) is used to select the next job in sequence and the process is repeated until all jobs are assigned. Makespan is computed for each of the n sequences developed and the sequence with minimum makespan is selected as the heuristic solution. In empirical tests, this heuristic performed slightly better than the Campbell, Dudek, and Smith procedure and appreciably better than a random procedure which will be discussed in section 2.5.5.

Nawaz, Enscore, and Ham [52] proposed a different heuristic approach. Working with the sum of the processing times for each job, they first select the two jobs with the greatest total processing times. The two jobs are ordered in a partial sequence that provides the best makespan for the partial sequence. The relative positions of the two jobs with respect to each other are fixed for the remaining steps of the procedure. The unscheduled job with the highest total processing time is tried in every possible position in the existing partial sequence, creating a new partial sequence with minimum makespan. This process is repeated until all jobs are assigned to sequence positions. Empirical testing by the authors and subsequent testing by Turner and Booth [68] indicate that, not only does this procedure produce better results on average than other known heuristics, it does so in less computer time than is required by Dannenbring's extensive search heuristic and only little more than is required for the Campbell, Dudek, and Smith approach.

Turner [71] suggested a modification to Dannenbring's extensive search heuristic that adds an all pairs exchange neighborhood search as a final stage. He found that results could be improved

significantly but at a substantial cost in additional computer time. He also modified the Nawaz, Enscore, and Ham procedure to add a final step. After determining an initial solution by the original procedure, each task is removed and reinserted at a different location. The best sequence is retained in each iteration and used as a starting solution for the next iteration. The procedure continues until no further inprovement is obtained. This modification proved to be the best procedure found but improvement came at a significant cost in computer processing time. These modified procedures can be classified as combinations of initial solution and neighborhood search procedures. The objective function in both cases is to minimize makespan.

A summary of the heuristics discussed above is given in Table I.

2.5.5 Other Solution Methods

Several unique solution methods that do not fall into one of the primary approaches discussed above have been proposed in the literature. Most of these appear only once or twice and receive no further attention. One of them, based on random sampling procedures from statistics, is mentioned more often. References to it can be found in Heller [39], Giglio and Wagner [32], King and Spachis [42], and Dannenbring [22]. In this procedure, a random sample from the (n!) permutation schedules is taken, the objective function is calculated for each sequence in the sample, and the sequence which provides the best value of the objective function is selected. This procedure is straight-forward and relatively quick. Its results, however, are not generally as good as the results of the more

TABLE I

SUMMARY OF HEURISTIC SOLUTION METHODS

Author	Heuristic	Classification
Page [53]	Merging	Single solution
	Pairing	Single solution
	Exchanging	Neighborhood search (Adjacent pairs)
Palmer [54]	Slope order index	Single solution
Petrov [58]	Johnson's rule	Single solution
Campbell, Dudek, and Smith [18]	Johnson's rule	Single solution
Gupta [37]	Job function	Single solution
Krone and Steiglitz [44]	Two stage search	Neighborhood search (Remove and reinsert) (All pairs)
Bonney and Gundry [13]	Slope Matching (no job waiting)	Single solution
Dannenbring [22]	Rapid access	Single solution
	with close order search (single pass)	Neighborhood search (Adjacent pairs)
	with extensive search (multiple pass)	Neighborhood search (Adjacent pairs)
King and Spachis [42]	Weighted job delay	Single solution
Nawaz, Enscore, and Ham [52]	Total job time	Single solution
Turner [71]	Modified Dannenbring extensive search	Combination
	Modified Nawaz, Enscore, and Ham	Combination

systematic approaches.

Ashour [3] [4] decomposes the job set into two or more parts of equal length. He then uses any method to solve each part for the best partial sequence and recombines the parts to get a total sequence. The procedure is repeated an unspecified number of times with differently partitioned subsets. The sequence yielding the best makespan is selected as the problem solution. This procedure has apparently been overtaken by more recently developed heuristics.

Axsater [5] proposed a dynamic programming approach to optimizing makespan in a flow shop where no job delay is allowed. Although this approach produces an optimal solution, it requires an excessive amount of computer processing time to achieve the optimum.

2.6 Other Factors Bearing on the Problem

To simplify the problem to one of manageable proportions, researchers in flow shop sequencing generally assume deterministic processing times. It is widely recognized, however, that such times are, in fact, stochastic. Muth [51] explored the effect of uncertainty in job times on optimal schedules. He concluded that schedule span (makespan) is not very sensitive to moderately large errors in job time estimates. He further found that the correlation ratio of job times had little effect on either average or minimum makespan unless the number of jobs was very large. Thus it would appear that the assumption of deterministic job times does not render research findings invalid for industrial application.

Processing times have, for the most part, been taken from a uniform distribution involving widely varying ranges. King and

Spachis [42] used two different Erlang distributions, one a low variance distribution with parameter k=9, and the other a high variance distribution with k=1. McMahon and Burton [50] used some job sets wherein processing times were correlated within jobs. Lagaweg et al. [45] also used correlated job times as well as some job sets that reflected either positive or negative trends over the machine sequence. In their survey of industry, Panwalkar et al. [52] found processing times showed similar trends for similar jobs. However, 63 percent of the respondents reported no positive or negative trend, but a similarly fluctuating pattern of job times on different machines. Implementing this pattern, Panwalkar and Kahn [56] used job sets wherein processing times were ordered on each machine. For example, the job that had the shortest processing time on machine one would also have the shortest times on all other machines. This case appears to be typical of the situation that would exist in cases of well planned group technology cells. Variations in processing times between jobs would occur primarily because of differences in lot sizes among jobs. Ignall and Schrage [40] show that, for the two-machine mean completion time problem and the three-machine makespan problem, changing location or scale of processing times will not change the optimal sequence. Although there is no formal proof to be found in the literature, this would indicate that the choice of distributional form for the processing times has very little impact on comparative results. Amar and Gupta [1] graphed the processing times and frequency of occurence from several real life problems and found no identifiable distributional pattern. This would indicate that any one distribution used in prior research was as valid as any other. A

summary of processing time distributions found in the literature together with their parameters is given in Table II.

Heller [39] notes that a flow shop has many different possible schedules but far fewer schedule times because several different schedules may produce the same makespan. He found that the distribution of schedule times could be reasonably described by a normal distribution. This result is essentially due to operation of the Central Limit Theorum for a single periodic Markov chain. This knowledge can be used to determine a sample size for the random sampling procedure that will reasonably ensure getting at least one sample from the lower tail of the distribution where the smallest values of makespan occur.

The literature dealing with the permutation flow shop contains a wide variety of problem sizes. Examples can be found ranging from very small, 3x2, problems to very large, 100x10 or 50x50. There is very little in the literature that specifically discusses this aspect of the problem. Amar and Gupta [1], in comparing simulated problems to those encountered in real life, discovered that the number of jobs for each machine is rarely as large as is given in some research simulations. They found that the ratio of jobs to machines, n/m, is rarely less than one or greater than four. This is due primarily to the need to maintain a smooth work flow. Ratios of less than one would result in very low machine utilization rates and high ratios would create a bottleneck which would not be permitted to persist. Further, while there are examples where an entire plant is one large flow shop with many machines or operations as is the case with process industries, it would appear that flow shop scheduling has its greatest

TABLE II

SUMMARY OF JOB TIME DISTRIBUTIONS

Author	Distribution	Parameters
Giglio and Wagner [32]	Uniform	1 - 30
Ashour [3] [4]	Uniform	1 - 30
McMahon and Burton [50]	Uniform	1 - 99
Baker [7] [8]	Uniform	1 - 99
Page [53]	Uniform	1 - 16
Campbell et al [18]	Uniform	1 - 99
Gupta [37]	Uniform	0 - 999
Krone and Steiglitz [44]	Uniform	0 - 1000
Bonney and Gundry [13]	Uniform	Not specified
King and Spachis [42]	Erlang (low var.) Erlang (high var.)	k = 9 k = 1

Other forms:

Job times correlated within jobs: Mc Mahon and Burton [50], Lagaweg et al. [45]

Job times with trend over machine sequence: Lagaweg et al. [45]

Ordered job times: Panwalkar and Kahn [56]

application potential in the manufacturing cells associated with group technology. This applicability was noted by Petrov [58] and by Bestwick and Hastings [12]. In such cases, the number of machines or operations is likely to be relatively moderate, estimated at no more than 20 to 25. Similarly the number of jobs to be scheduled through the cell at any scheduling cycle is likely to be relatively small, estimated at 4 to 12. The very nature of group technology would seem to indicate that the number of products or components which were of sufficient similarity to be assigned to a single cell for processing would not run to very many. Even as early as 1971, responses to an industry survey conducted by Panwalkar et al. [55] indicated that nearly 20 percent rarely scheduled more than 10 jobs on 10 machines.

A final consideration that has plagued flow shop researchers for some time is that of the practical applicability of their research results. As is evident from the earlier discussions in this chapter, much research effort has been devoted to shop scheduling. These discussions have touched only on the flow shop case. Yet there is little to indicate that any of the several approaches to problem solution have been widely adopted in industrial practice. Pounds [59] discusses this phenomenon at some length based upon his work with industrial scheduling personnel. He found that very few schedulers recognized a need for improved scheduling methods because there were few apparent problems with existing methods. It was a clear case of 'if it isn't broke, don't fix it'. Looking further into the situation, he found that other functions, such as marketing and production, were taking actions unknown to schedulers which were intended to alleviate scheduling problems. Marketing would resist

short delivery dates or production would run overtime in order to avoid missing promised deliveries. In some cases, management purchased additional production equipment to alleviate scheduling problems. Although these findings occurred more than twenty years ago, it is unlikely that the situation has changed much in the intervening years. It seems, then, that researchers must convince management that there is room for improvement in the scheduling process. With the current state of international competition and the drive to improve productivity, the time would seem ripe to reap the benefits of even small gains in productivity that might result from improved scheduling methods.

CHAPTER III

RESEARCH METHODOLOGY

3.1 General Approach

Neighborhood search procedures provide a systematic method of seeking to improve the solutions to a wide range of combinatorial problems. While they are capable of achieving a locally optimal solution, they do not guarantee that the solution is globally optimal over the entire solution space. Baker [6] describes three steps of a neighborhood search procedure as follows:

> Step 1: Obtain a sequence to be an initial seed and evaluate it with respect to the given measure of performance.
> Step 2: Generate and evaluate all the sequences in the neighborhood of the seed. If none of the sequences are better than the seed with respect to the given measure of performance, stop. Otherwise proceed.
> Step 3: Select one of the sequences in the neighborhood that improved the measure. Let this sequence be the new seed. Return to step 2. (p. 67)

Within this procedural framework, the analyst must still specify a method of obtaining the initial seed, a specific neighborhood generating mechanism, and a method of selecting the sequence to be the new seed. This study is concerned with all three of these specifications and will analyze the options with respect to both the general results achieved and the time required to achieve them. Our problem will require determining the best combination of the options available to the analyst.

The general approach to be used in this study involves computer simulation of the flow shop in which solutions are limited to permutation schedules. Baker [6] and Conway et al. [20], among others, have shown that, except for specially constructed flow shop problems, a permutation schedule provides a solution that is either optimal or so close to optimal that the additional computational effort necessary to pursue non-permutation schedules is not cost effective. Solution algorithms will be coded in Fortran (see appendix) and run on the IBM 3081K available through the Oklahoma State University Computer Center.

The problem set to be utilized to provide the data for analysis will consist of a series of flow shop problems with randomly generated processing times. Problems will be generated from a range of problem sizes in order to provide some limited capability to generalize the analytical results. The problem set will have integer processing times generated from a uniform (0,99) distribution as has been used in much of the previous flow shop research (See table II in Chapter 2). The job set to be generated is summarized in Table III. The rationale for selecting these problem sizes and number of replications of each problem size is discussed in section 3.2. The exploration of the comparative performance of heuristics and neighborhood search procedures on problems with processing times correlated across machines for each job is left for a follow-on study to this one.

Although it would be desirable to have an optimal solution to each problem as a standard against which to measure the performance of the heuristics and neighborhood search procedures, initial attempts to find optimal solutions indicate that the computer processing time

TABLE III

SUMMARY OF PROBLEMS IN THE JOB SET

N \ M	4	8	12	16
4	10	10	10	10
8	10	10	10	10
12	10	10	10	10
16	10	10	10	10

required is prohibitive. French's [28] mixed integer linear programming model, couched in terms of the notation given by Baker [6], was used to formulate an MPSX model for execution on the IBM 3081K. With only a few exceptions, optimal solutions for problems with four and eight jobs were readily obtained. The same is true for the 12 job by four machine problems. For other problems in the problem set, 90 minutes of computer processing time was insufficient to obtain optimal solutions. In many cases, an integer solution was found but there was not sufficient time to determine whether this integer solution was optimal. Therefore, only limited conclusions can be drawn with regard to the ability of the heuristics and neighborhood search procedures to approach optimality. As a result of this limitation, the best heuristic solution for each problem in the problem set and the time required to obtain that solution will be used as basis against which to compare the performance of the heuristics and search routines.

3.2 Experimental Design

The general research design is a two-phased, full factorial design. The first phase is intended to provide answers to the question of which heuristic is best as a stand alone procedure. The factors included in this phase are number of jobs (N), number of machines (M), and the heuristic procedures (H). The second phase is intended to answer questions concerning the neighborhood search procedures in combination with the heuristics as initialization procedures. Phase two includes the factors in phase one plus the search procedures and improvement rules to be employed. The levels of

each factor and the rationale for choosing them are discussed in the subsections which follow.

3.2.1 Number of Jobs

The levels chosen for this factor are 4, 8, 12, and 16 as shown in Table III. These levels are intended to be representative of the levels that might reasonably be found in industry. Although the literature is rife with research involving many more jobs (typically up to 50), Amar and Gupta [1] noted that industry rarely schedules the number of jobs given in many research simulations. Thus the highest level chosen, 16, is an attempt to provide a more realistic maximum. Level one is chosen with a value greater than three so that no optimizing heuristic is available. Levels two and three evenly span the range between levels one and four.

3.2.2 Number of Machines

The levels chosen for this factor are also 4, 8, 12, and 16 as shown in Table III. These levels were chosen for testing based upon findings by Amar and Gupta [1] that the ratio of n to m is rarely less than one and rarely greater than four. Although n to m ratios less than one are included (i.e., four jobs on 16 machines), the findings with respect to the low ratio combinations may provide some insight as to the ability to generalize the results.

3.2.3 Initial Solution Heuristics

The heuristics that have been previously proposed in the literature are discussed in chapter 2 and are summarized there in

table I. For purposes of this study, only those heuristics that can be applied in situations which permit in-process inventory will be considered. Although previous studies have compared certain aspects of these heuristics, the research objectives were somewhat different than those sought in this study. In a 1981 master's thesis, Park [57] compared several heuristics without distinguishing between heuristics that produced single initial solutions and those that employed a neighborhood search technique to improve upon a starting solution sequence. A similar study by Setiaputra [64] also failed to make this distinction. Dannenbring [22] noted this distinction in his analysis of results but was seeking totally different research objectives. These previous studies give rise to certain expectations as to the outcome of selected research questions in the current study. Nevertheless, a wide range of existing heuristics will be studied here. It is possible that one of the lesser performing heuristics will provide the best seed sequence for the subsequent neighborhood search procedures. However, in order to keep the study within manageable size, only one heuristic of those using similar approaches is included. For example, Palmer [54] and Gupta [37] employ very similar appoaches and previous research has indicated that Gupta's model gives better results in general. Therefore, only Gupta's heuristic is included here. Of the heuristics producing a single initial solution, as noted in Table I, those selected for inclusion in the present study are given in Table IV together with the abbreviations by which they will be identified throughout this study.

The random sampling approach is included here for the same reason it has been included in other studies. This method provides a

TABLE IV

HEURISTICS INCLUDED IN THE STUDY

Author(s)	Approach	Mnemonic
Petrov	Johnson's Rule Single pass	PTV
Campbell, Dudek, and Smith	Johnson's Rule Multiple Pass	CDS
Gupta	Job Function	GTA
Dannenbring	Rapid Access Weighted Slope Function	DRA
Nawaz, Enscore, and Ham	Total Job Time	NEH
-----	Random Sampling	RDM

sub-optimal solution for a relatively small expenditure of computer time. Thus it can serve as a benchmark for other methods, particularly with respect to computational effort. It should be noted, however, that sample sizes for random sampling are arbitrarily chosen. The sample size, N, selected for each value of n (number of jobs) takes into account the desire to include an adequate number of the n! possible permutations as well as the practical factor of processing time limitations. The sample sizes chosen are patterned after those used by Dannenbring [22] and are given in Table V together with the value of n!.

For phase two of the research, a seventh initializing procedure will be included as an additional level of this factor. A simple ordinal sequence (i.e., 1-2-3-4-etc.) can be used to initialize the neighborhood search process. This sequence can be produced with zero processing time. It is possible that application of a neighborhood search procedure to the ordinal sequence can produce good results in less time than some combinations of initializing heuristics and neighborhood search procedures. If such is the case, one can dispense with the initializing heuristics altogether and employ only neighborhood search procedures on some arbitrarily chosen initial sequence of jobs.

3.2.4 Neighborhood Search Procedures

There are any number of neighborhood generation schemes which might be employed. Baker [6] and Dannenbring [22] mention several specifically, as do other authors. Our purpose in selecting neighborhood generation schemes to be used in this study is to span

TABLE V

SAMPLE SIZES FOR RANDOM SAMPLING

n	N	n!	%
4	10	24	.4167
8	400	40,320	.00992
12	1500	4.79×10^{8}	3.13×10^{-6}
16	2000	2.0923×10^{13}	9.5589×10^{-11}

the range from very simple to relatively complex and to provide a broad sample of sizes of the neighborhood generated. The generation schemes to be employed in this study are discussed below and are summarized in Table VI. Some of these are frequently found in the literature. Others, thought to be original, are logical extensions of schemes found in the literature.

One generation scheme frequently mentioned in the literature is adjacent pair switching. This scheme was employed in Dannenbring's close order search and extensive search heuristics. In this scheme, the neighborhood is created by exchanging positions of two adjacent jobs. The neighborhood generated has size n-1. For example, with n=3 and original sequence 123, the two sequences produced by this scheme would be 213 and 132.

A logical outgrowth of adjacent pair switching is to extend the switching to all pairs. When switching job i with job j, the redundancy in the resulting neighborhood can be eliminated by placing restrictions on the value of j. By specifying the scheme as exchange all i and j for i = 1, 2,..., n-1 and j = i+1, i+2,..., n, the redundant sequences will not be generated and the resulting neighborhood will have size n(n-1)/2. Using the previous example with n=3 and seed sequence 123, this scheme would produce a neighborhood of 213, 321, and 132.

Still another possible scheme involves switching adjacent doublets. Every possible set of four adjacent jobs has the first two jobs switched with the last two. This scheme produces a neighborhood of size n-3. For a five job problem with seed sequence 12345, the neighborhood consists of sequences 34125 and 14523.

TABLE VI

NEIGHBORHOOD GENERATION SCHEMES
INCLUDED IN THE STUDY

Mnemonic	Generation Scheme	Neighborhood Size
ADJP	Adjacent pair switching	n-1
ALLP	All pairs switching	$n(n-1)/2$[(a)]
ISGL	Remove single job and reinsert in all possible positions	n(n-1)
ADJD	Adjacent doublet switching	n-3
IAJP	Remove adjacent pair and reinsert as pair in all possible positions	(n-1)(n-2)
IALP	Remove all pairs and reinsert as pair in all possible positions	$n(n-1)^2$

(a) Actual neighborhood size is n(n-1) but exactly half of the sequences generated are redundant. The generation scheme can be written in such a way that redundant sequences are not generated.

Another scheme is similar to the sequence building procedure employed by Nawaz, Enscore, and Ham [49]. Each job is removed in turn from the sequence and reinserted at all possible positions to create new sequences. This scheme produces a limited number of redundant sequences but modifying the generation algorithm to avoid redundancy generally is more difficult and takes more time than simply calculating the objective function more than once for the redundant sequences. The size of the neighborhood is n(n-1). For the three job example, we get a neighborhood of 213, 231, 213, 132, 312, and 132.

We might extend the removal and reinsertion of a single job to removing an adjacent pair of jobs and reinserting them as a pair in every other possible position in the sequence. This scheme produces a neighborhood of size (n-1)(n-2). In a four job problem with seed sequence 1234, removing 12 generates sequences 3124 and 3412, removing 23 generates sequences 2314 and 1423, and removing 34 generates 3412 and 1342. As was the case with removal and reinsertion of single jobs, this scheme will generate some redundant sequences with larger values of n. Again it is quicker to simply calculate the objective function more than once for the redundant sequences than to modify the algorithm to eliminate them.

The preceding scheme can be extended to the removal and reinsertion as a pair of all possible pairs. This procedure will generate a much larger neighborhood than other schemes discussed here. The neighborhood size is $n(n-1)$. It will, however, also generate a larger number of redundant sequences. For example, for a four job problem, 36 sequences are generated of which 16 are redundant and, for a five job problem, 80 sequences are generated of which 33 are

redundant. Of the 120 possible sequences for the five job problem, 72 of them do not appear at all in this neighborhood.

Many other extensions or perturbations of the schemes previously discussed could be devised. One would expect that, as the neighborhood size increases, the probability of the neighborhood including an optimal sequence would also increase. However, increasing the complexity of the scheme to produce larger neighborhoods also increases dramatically the computer time required to generate the neighborhood and, perhaps more importantly, the time to compute the objective function values associated with the sequences in the larger neighborhoods. In order to keep this study within the bounds of practicality, the last generating scheme discussed above will be used to generate the largest neighborhoods for the study. The range of neighborhood sizes, from n-3 to $n(n-1)^2$, should provide an indication of the impact of neighborhood size adequate to permit some generalizing of the results.

In phase two of this research, the heuristics serve as initializing procedures for the various neighborhood search routines discussed above. Since our research interest is in the combined results or interactions between the heuristics and search routines, these two factors will be combined and each of the 42 (seven initializing heuristics and six search routines) combinations will be identified as a level of the combined factor. This will enable us to treat the interaction as a main effect in phase two.

3.2.5 Improvement Rules

Two basic approaches for selecting the sequence to seed the next

iteration of a neighborhood search procedure may be used. One may search sequentially through a given neighborhood until an improved sequence is found and use this improved sequence to seed the next iteration. This approach is referred to as the "first improvement" rule and represents one level of this factor. One may also search the entire neighborhood and select the sequence which provides the greatest improvement in the objective criterion as the seed for the subsequent iteration. This approach is the "best improvement" rule and constitutes the second level of this factor.

The first improvement rule will likely require more, but shorter, iterations. The best improvement rule will likely reach the local optimum in fewer iterations but each iteration will require more computer processing time. Analysis of the main effects of this factor should provide some indication of which of these approaches, if either, is better on average.

3.2.6 Replication

Because the processing times for a problem of a given size (n x m) are randomly generated, it would appear that heuristics and search procedures should be tested against more than one problem of each size. This will tend to provide a better estimate of performance because results will not be biased by the peculiarities of a single randomly generated problem.

Most statistical texts (see Winer [76], for example) provide formulas for determining an appropriate sample size or number of replications. Such formulas are dependent upon establishing a minimum difference which is desired to be detected as well as the acceptable

levels of Type I and Type II error. Since this study is very much exploratory in nature and there is no precedence in the literature concerning the differences which might be expected in the proposed performance measures, efforts to compute a sample size would be futile at this point. However, review of Table 3.13-1, page 223, of Winer [76] indicates that 10 problems of each size should be sufficient to provide tests of adequate power. If initial analysis of the data indicates no significant differences, then a larger sample will need to be taken.

3.3 Measures of Performance

Analysis of the performance of the heuristics and neighborhood search procedures requires that some measure of this performance be defined. A heuristic, with or without augmentation by a neighborhood search procedure, produces a processing sequence which results in a determinable objective function value. Based upon the discussion of optimization criteria in section 2.4, performance will be evaluated on the basis of optimizing (minimizing) total processing time or makespan. Performance measures to be employed in this study can be divided into two general categories: comparative measures and achievement measures. These measures are discussed in the sub-sections which follow and are summarized in Table VII.

3.3.1 Comparative Measures.

Statistical comparison of performance can be done parametrically or non-parametrically. For a non-parametric comparison, one need only rank the makespans of the heuristics or heuristic and search routine

TABLE VII

SUMMARY OF PERFORMANCE MEASURES

Comparative Measures:

$$SE = MS/MS^*$$

$$CE = T/T^*$$

Achievement Measures:

$$H_j = X_j/N \quad \text{for } j = 0, 1, 3, 5$$

Legend:

- SE = solution efficiency
- CE = computational efficiency
- MS* = makespan of best solution
- MS = heuristic makespan
- T = heuristic, search routine, or combined processing time
- T* = processing time of best solution
- N = number of problems considered
- H_j = proportion of times solution within j percent of best heuristic solution
- X_j = number of times solution within j percent of best heuristic solution

combinations for each problem. The resulting ranks can then be subjected to a standard analysis of variance procedure (See Conover [19]).

To give some consideration to processing times, the ranking procedure can be modified so that ties in makespan can be broken with computer processing times. This procedure produces a time adjusted ranking which can then be subjected to an analysis of variance.

Non-parametric analysis is relatively simple to perform but does not provide as complete an analysis as is possible with parametric procedures. It is impossible to assess the effect of number of jobs (n) or number of machines (m) with the non-parametric procedure because the average ranks for each level of these factors will be identical. This procedure does enable us to assess the effect of the heuristics and the heuristic/search routine combinations which are the primary concern of this study. However, because of the limitations on the analysis of the effects of other factors, the primary analysis will be done by parametric methods.

Use of parametric analysis requires further designation of comparative performance measures. There are two distinct aspects of performance that are of interest in this study.

The first aspect for any solution is how close the resulting makespan comes to the best solution. A number of measures can be found in the literature for comparing the performance of one heuristic with another. In cases where an optimal solution is known or can be estimated, heuristics can be compared on the basis of relative error. Dannenbring [21] [22] uses this factor as one comparative measure. Park [57] uses an average makespan to which he applies a multiple

comparison technique devised by Dunnett [26] which compares the mean of the experimental populations with the mean of a control or standard population. Setiaputra [64] transforms the makespan results into rankings and uses Friedman's well-known non-parametric test to determine if there are significant differences in the rankings. It is also possible to quantify the proximity to a best solution to permit the direct application of a parametric procedure. This is simply the complement of Dannenbring's error ratio. Such a measure, call it solution efficiency, permits homogenation of the results of problems with widely varying makespans. This measure will be used in this study to provide a measure of proximity to the best solution. Solution efficiency is computed as:

$$SE = MS/MS^* \tag{3.1}$$

where MS* is the best makespan for each problem and MS is the heuristic makespan. Values of this performance measure will be greater than or equal to one with smaller values indicating better performance. SE will be computed for each problem in the job set and then averaged as appropriate to provide data for statistical analysis.

The second aspect of the solution that is of interest is how long it takes to obtain the solution. Computer processing time can be measured directly but this measurement has no meaning in and of itself. Although there is frequent reference in flow shop literature to the direct comparison of computer processing times between heuristic methods, there can be found no single performance measure that gives the ability to jointly compare the processing time and the goodness of the solution obtained. Hierarchical analysis, as proposed by Saaty [62], offers some promise in this area which may be explored

as a follow up to this study. We can, however, compare the heuristic and/or search procedure processing time to the time that was required to obtain the optimal solution. We, therefore, propose a measure that we shall call computational efficiency. This is a measure of relative efficiency of the heuristic procedure compared to the best solution achieved. Computational efficiency is calculated as:

$$CE = T/T^* \tag{3.2}$$

where T is the heuristc processing time in milliseconds and T* is the processing time required to achieve the best solution. Values of this performance measure will be greater than zero and smaller values are indicative of greater heuristic computational efficiency. As was the case with SE above, CE will be calculated for each problem in the job set and then aggregated appropriately for the statistical analysis.

Although the exact distributions of SE and CE are unknown, this fact should have little impact on the validity of the statistical ANOVA procedures applied to these measures. Kleijnen [40] cites findings by Scheffe [65] which indicate non-normality has little effect on the power of the F-test when the number of degrees of freedom is large and unequal variances have little effect when the number of observations per cell is equal. Donaldson [24] finds similar results in empirical tests of a single factor experiment with an equal number of observations per level.

3.3.2 Achievement Measures.

Achievement measures of performance have been used in a number of previous studies. This measure is a proportional measure of heuristic achievement in that it reflects the proportion of times that the

heuristic solution either achieves or comes within a specified range of the optimal or best heuristic solution. Setiaputra [64] measured the proportion of times that the heuristic solution was within five percent of the best heuristic solution. Park [57] used a similar measure with a range of one percent. Dannenbring [21] [22] used a measure of the proportion of times that a heuristic solution achieved the actual or estimated optimal. Dannenbring [21] points out, however, that his measure is only meaningful when combined with a measure of solution efficiency. For example, a heuristic that achieved the optimal 80 percent of the time but produced very poor solutions other times would be less desireable than one which achieved the optimal only 60 percent of the time but was very close to optimal other times. Yet a measure of the proportion of optimal solutions would favor the first heuristic. It would seem prudent, therefore, to use more than one achievement measure to provide a better assessment of the performance of a given heuristic. In view of this and the impracticality of obtaining optimal solutions, we will adopt a series of achievement ratings that, taken together, will indicate the range of achievement with respect to the best heuristic solution found. These measures will be computed as:

$$H_j = X_j/N, \quad j = 0, 1, 3, 5 \tag{3.3}$$

where H_j = proportion of times solution within j percent of best solution
X_j = number of times solution within j percent of best solution
N = number of problems considered

3.4 Research Hypotheses

The research questions address three main issues: (1) the effectiveness of the initial solution heuristics as stand-alone procedures; (2) the effectiveness of the neighborhood search procedures in improving on initial sequences; and (3) which of the two improvement rules is more efficient. These issues lead to the formulation of a series of hypotheses which are given below together with the rationale underlying each one.

3.4.1 Phase One Hypotheses

Phase one of the research design addresses the effectiveness of initial solution heuristics as stand-alone procedures. Three factors are involved in this phase. Although our primary interest involves the main effects due to the heuristic procedures, it is also necessary to check the main effects of both number of jobs and number of machines as well as certain of the interactions between factors.

The following hypotheses will be tested during phase one:

Heuristic Main Effects: In a flow shop typified by a given set of operating conditions under study, there is no significant difference among the six heuristics in terms of either solution efficiency or computational efficiency.

N Main Effects: There is no significant difference among four levels of numbers of jobs in terms of either solution efficiency or computational efficiency.

M Main Effects: There is no significant difference among four levels of number of machines in terms of either solution

efficiency or computational efficiency.

Of less interest in this research are the interaction terms of the model. There are three two-way interactions and one three-way interaction to be included in the model for this phase. Each of these would have a null hypothesis which states that the interaction is not significant in terms of either solution efficiency or computational efficiency. Although not of primary interest, these interactions, if significant, can be of interest in evaluating the performance of the heuristics. Particularly the job size (n x m) interaction may be helpful in selecting an appropriate heuristic in practical scheduling situations.

If significant differences are found with respect to the main effects, it will be necessary to apply one of numerous multiple comparison procedures (MCP) to determine which levels of the factor differ. The Statistical Analysis System (SAS), which will be used to analize the data, provides several options for multiple comparison procedures. Among these are procedures attributed to Ryan [61], Einot [27], Gabriel [29], and Welsch [75] which control the experiment-wise error rate. The SAS User's Guide [63] notes that these procedures appear to be among the most powerful step down multiple stage tests in current literature. Their F-test has the advantage of being compatible with the overall ANOVA F-test in that it rejects the complete null hypothesis only if the overall F-test does so. Use of a preliminary F-test decreases the power of all other multiple comparison methods available in SAS except for Scheffe's test.

3.4.2 Phase Two Hypotheses

Phase two of the research addresses several aspects of the effectiveness of neighborhood search procedures. Four factors are involved in this phase. Our primary interest lies in the main effects due to the various combinations of initializing heuristics and search routines. We are also interested in the effect of the improvement rules as well as the effects of both number of jobs and number of machines and the interaction effects.

The following hypotheses will be tested during this phase:

Combination Main Effects: In a flow shop typified by a particular set of operating conditions under study, there is no significant difference among the 42 combinations of initializing heuristics and neighborhood search routines in terms of either solution efficiency or computational efficiency.

Improvement Rule Main Effects: There is no significant difference between the first improvement and best improvement rules in terms of either solution efficiency or computational efficiency.

N Main Effects: There is no significant difference among the four levels of number of jobs in terms of either solution efficiency or computational efficiency.

M Main Effects: There is no significant difference among the four levels of number of machines in terms of either solution efficiency or computational efficiency.

Of less interest in this study is the effect of the interaction

terms of the model. There are six two-way interactions, four three-way interactions, and one four-way interaction to be included in the model for this phase. Each of these would have a null hypothesis which states that the interaction is not significant in terms of either solution efficiency or computational efficiency.

As indicated, these interactions do not represent the primary focus of the research. Nevertheless, certain of these interactions, if significant, can be of some interest in evaluating the performance of the heuristic/search routine combinations. In particular, the job size (n x m) interaction with the combinations and with the improvement rules may be helpful in selecting appropriate parameters for neighborhood searches in practical applications.

CHAPTER IV

ANALYSIS OF THE DATA

4.1 Phase One

This phase is concerned with which of the heuristic methods is best as a stand alone procedure. Each of the six heuristics was applied to each of the 160 problems in the problem set producing a total of 960 solution sequences. The makespan was calculated for each sequence and the computer processing time required to achieve each solution, measured in milliseconds, was recorded.

4.1.1 Analysis of Comparative Performance Measures

The best solution for each problem was identified together with the computer processing time required to produce it. If more than one heuristic achieved the shortest makespan, the one with the shortest processing time was chosen as the best heuristic solution.

Solution efficiency (SE) was calculated for each solution using equation 3.1. Computational efficiency (CE) was also calculated using equation 3.2. The resulting values were then subjected to an analysis of variance using the following models:

SE = N M H N*M N*H M*H N*M*H

CE = N M H N*M N*H M*H N*M*H

The results of the analysis of variance on solution efficiency

are given in Table VIII. All of the main effects proved to be significant as did all of the two-way interactions. Only the three-way interaction was not significant. We would reject all of the phase one hypotheses in terms of SE except for the one concerning the three-way interaction. The existence of significant interaction effects makes the interpretation of the main effects of the model much more difficult if not impossible. This is discussed in greater detail in Section 4.1.2 below.

Having found the main effects to be significant, the Ryan, Einot, Gabriel, Welsch F-test (REGWF) multiple comparison procedure (MCP) was applied to determine which levels of the factors were significantly different. These results are given in Tables IX, X, and XI. Recalling that smaller values of SE are preferred, Table IX shows that the solution efficiency decreases as the number of jobs increases with no significant difference between 8 and 12 jobs. Table X shows a similar relationship between SE and the number of machines with no significant differences among the three higher levels of this factor. Table XI reflects our primary concern in this phase. The Nawaz, Enscore, and Ham (NEH) heuristic produces the best results followed by CDS and the random (RDM) heuristic which do not differ significantly from each other. Dannenbring's rapid access (DRA) procedure is a distant fourth, followed by Petrov's (PTV) procedure, and Gupta's (GTA) heuristic is in last place.

The analysis of variance results for computational efficiency are given in Table XII. All of the main effects except number of machines are significant as are all of the interactions except for the interaction between number of machines and heuristics. A summary of

TABLE VIII

ANOVA TABLE FOR PHASE ONE VARIABLE SE

DEPENDENT VARIABLE: SE

SOURCE	DF	SUM OF SQUARES	MEAN SQUARE
MODEL	95	1.41409113	0.01488517
ERROR	864	1.21587978	0.00140727
CORRECTED TOTAL	959	2.62997091	

F VALUE	PR > F	R-SQUARE	C.V.
10.58	0.0	0.537683	3.5794
	ROOT MSE		SE MEAN
	0.03751357		1.04804650

SOURCE	DF	ANOVA SS	F VALUE	PR > F
N	3	0.16151724	38.26	0.0001
M	3	0.03516353	8.33	0.0001
H	5	0.92834839	131.94	0.0
N*M	9	0.06033726	4.80	0.0001
N*H	15	0.10657247	5.05	0.0001
M*H	15	0.07537735	3.57	0.0001
N*M*H	45	0.04627489	0.73	0.9062

TABLE IX

MCP FOR MAIN EFFECTS OF N ON PHASE ONE VARIABLE SE

SAS

ANALYSIS OF VARIANCE PROCEDURE

RYAN-EINOT-GABRIEL-WELSCH MULTIPLE F TEST FOR VARIABLE: SE
NOTE: THIS TEST CONTROLS THE TYPE I EXPERIMENTWISE ERROR RATE

ALPHA=0.05 DF=864 MSE=.0014073

NUMBER OF MEANS	2	3	4
CRITICAL F	5.01924	3.00614	2.61521

MEANS WITH THE SAME LETTER ARE NOT SIGNIFICANTLY DIFFERENT.

REGWF GROUPING	MEAN	N	N
A	1.062470	240	16
B	1.053923	240	12
B	1.048488	240	8
C	1.027305	240	4

TABLE X

MCP FOR MAIN EFFECTS OF M ON PHASE ONE VARIABLE SE

SAS

ANALYSIS OF VARIANCE PROCEDURE

RYAN-EINOT-GABRIEL-WELSCH MULTIPLE F TEST FOR VARIABLE: SE
NOTE: THIS TEST CONTROLS THE TYPE I EXPERIMENTWISE ERROR RATE

ALPHA=0.05 DF=864 MSE=.0014073

NUMBER OF MEANS	2	3	4
CRITICAL F	5.01924	3.00614	2.61521

MEANS WITH THE SAME LETTER ARE NOT SIGNIFICANTLY DIFFERENT.

REGWF GROUPING	MEAN	N	M
A	1.053840	240	16
A A			
A	1.051290	240	12
A A			
A	1.049078	240	8
B	1.037978	240	4

TABLE XI

MCP FOR MAIN EFFECTS OF H ON PHASE ONE VARIABLE SE

SAS

ANALYSIS OF VARIANCE PROCEDURE

RYAN-EINOT-GABRIEL-WELSCH MULTIPLE F TEST FOR VARIABLE: SE
NOTE: THIS TEST CONTROLS THE TYPE I EXPERIMENTWISE ERROR RATE

ALPHA=0.05 DF=864 MSE=.0014073

NUMBER OF MEANS	2	3	4	5	6
CRITICAL F	5.72346	3.69182	2.91163	2.38224	2.22447

MEANS WITH THE SAME LETTER ARE NOT SIGNIFICANTLY DIFFERENT.

REGWF GROUPING	MEAN	N	H	
A	1.097893	160	3	(GTA)
B	1.069816	160	1	(PTV)
C	1.057942	160	4	(DRA)
D	1.031421	160	6	(RDM)
D	1.029096	160	2	(CDS)
E	1.002111	160	5	(NEH)

TABLE XII

ANOVA TABLE FOR PHASE ONE VARIABLE CE

DEPENDENT VARIABLE: CE

SOURCE	DF	SUM OF SQUARES	MEAN SQUARE
MODEL	95	8443608.05406504	88880.08477963
ERROR	864	9708175.03625546	11236.31369937
CORRECTED TOTAL	959	18151783.09032051	

F VALUE	PR > F	R-SQUARE	C.V.
7.91	0.0	0.465167	339.5592
	ROOT MSE		CE MEAN
	106.00147970		31.21737796

SOURCE	DF	ANOVA SS	F VALUE	PR > F
N	3	507063.97360945	15.04	0.0001
M	3	61064.86941375	1.81	0.1435
H	5	2971638.73915726	52.89	0.0
N*M	9	310561.05392157	3.07	0.0012
N*H	15	3446003.49940589	20.45	0.0
M*H	15	159431.19010737	0.95	0.5120
N*M*H	45	987844.72844976	1.95	0.0002

actions with respect to phase one hypotheses is contained in Table XIII.

Again, having found two of the main effects to be significant, the REGWF procedure was applied to determine the differences among levels for these factors. These results appear in Tables XIV and XV. Table XIV shows that the average computational efficiency for n=4 differs significantly from that of the other three levels for this factor. Table XV shows that the computational efficiency of PTV, DRA, GTA, and CDS all have average values less than one and do not differ significantly from each other. RDM is next in desirability and NEH is a distant last in this measure of performance.

4.1.2 Interpretation of the Results

The presence of significant interactions creates some difficulty in interpreting the main effects of the model. A closer examination of the interaction effects is in order before attempting such an interpretation. Graphical plots of each of the significant interactions were made. With respect to SE, the n x m interaction indicates that there is some varying effect. For example, at n=4, the ordering of results from best to worst was m=12, m=16, m=8, and m=4. At levels n=8 and n=12, the order was m=4, m=8, m=12, and m=16. At level n=16, the order was m=4, m=12, m=8, and m=16. Similarly, at m=4, the ordered results were n=12, n=4, n=8, and n=16. At all other levels of m, the order was n=4, n=8, n=12, and n=16. Thus, it would appear that, although there is some confounding of the main effects due to significance of the interactions, some very general tendencies are still evident. One finds somewhat similar results when one

TABLE XIII

SUMMARY OF ACTIONS FOR PHASE ONE HYPOTHESES

HYPOTHESIS: There is no significant difference in SE (CE) due to:	SE	CE
N	Reject	Reject
M	Reject	Accept
H	Reject	Reject
N*M	Reject	Reject
N*H	Reject	Reject
M*H	Reject	Accept
N*M*H	Accept	Reject

TABLE XIV

MCP FOR MAIN EFFECTS OF N ON PHASE ONE VARIABLE CE

SAS

ANALYSIS OF VARIANCE PROCEDURE

RYAN-EINOT-GABRIEL-WELSCH MULTIPLE F TEST FOR VARIABLE: CE
NOTE: THIS TEST CONTROLS THE TYPE I EXPERIMENTWISE ERROR RATE

ALPHA=0.05 DF=864 MSE=11236.3

NUMBER OF MEANS	2	3	4
CRITICAL F	5.01924	3.00614	2.61521

MEANS WITH THE SAME LETTER ARE NOT SIGNIFICANTLY DIFFERENT.

REGWF GROUPING	MEAN	N	N
A	70.982	240	4
B	18.865	240	12
B	18.775	240	16
B	16.247	240	8

TABLE XV

MCP FOR MAIN EFFECTS OF H ON PHASE ONE VARIABLE CE

SAS

ANALYSIS OF VARIANCE PROCEDURE

RYAN-EINOT-GABRIEL-WELSCH MULTIPLE F TEST FOR VARIABLE: CE
NOTE: THIS TEST CONTROLS THE TYPE I EXPERIMENTWISE ERROR RATE

ALPHA=0.05 DF=864 MSE=11236.3

NUMBER OF MEANS	2	3	4	5	6
CRITICAL F	5.72346	3.69182	2.91163	2.38224	2.22447

MEANS WITH THE SAME LETTER ARE NOT SIGNIFICANTLY DIFFERENT.

REGWF GROUPING	MEAN	N	H
A	152.69	160	5 (NEH)
B	33.26	160	6 (RDM)
C	0.70	160	2 (CDS)
C			
C	0.22	160	3 (GTA)
C			
C	0.22	160	4 (DRA)
C			
C	0.22	160	1 (PTV)

examines the n x h and m x h interactions pertaining to SE. With some minor exceptions wherein crossovers do occur, the ordering of the heuristics at different levels of n and m are consistent. The resulting lines on the graph are not parallel, but this is indicative of the significance of the interaction effect. The model cannot be used to predict performance with respect to SE at various levels of the factors, but prediction was not our purpose. Thus, it is felt that main effect tendencies, albeit very general in nature, can be observed and reported.

The differences in SE due to the main effects of n indicate that those heuristics which do not produce the best heuristic results tend to miss the mark by a wider margin as the number of jobs increases. This result is not unexpected in that a problem with few jobs has fewer sequences in the set of n! possible sequences than does a problem with more jobs to be scheduled. It is more likely that any of the heuristics will find an optimal or near optimal sequence under conditions of small n. As n increases, n! increases at an increasing rate, so that a given heuristic is less likely to hit upon an optimal or near optimal sequence from among the much larger set of possible sequences.

The differences in SE due to the main effects of m show a similar tendency as for the factor n above in that the average values of SE increase as m increases. The logic of this relationship is not as readily apparent as was the case with n because m has no direct bearing on the number of potential sequences. The number of machines, nevertheless, is a factor contributing to the complexity of the problem. Evidence of this was seen in the computer processing times

associated with the attempts to find optimal solutions. It is also evident in the achievement measures to be discussed in the next section. In the case of the optimization procedures, with fixed values of n, the time required to obtain an optimal solution increased as m increased. In the case of the achievement measures, we can see a general tendency among all heuristics to produce poorer results as m increases. While this evidence does not "explain" the complicating influence of the number of machines, it does provide testimony of its presence.

The differences in SE due to the main effects of the heuristics are very much as expected based upon the review of previous research. The NEH heuristic retains the best partial sequence as each job is inserted into the previous partial sequence in what is, in effect, a trial and error approach. Because this is, by far, the most complex of the heuristic procedures, one might expect that it would produce, on average, the best results. The well known CDS procedure creates a number of sub-problems and then retains the sequence that provides the best solution. Compared to other heuristics that create only one solution sequence, it should be expected to produce better results. The relatively good performance of RDM is somewhat surprising despite the fact that Dannenbring [21] obtained similar results. But again this procedure chooses the best of a series of sequences, so it might be expected to outperform heuristics which generate only a single sequence. Dannenbring's rapid access was not intended to be a stand alone procedure. Rather it was designed as an initializing procedure to provide a starting sequence for additional search procedures. That it can produce average results within six percent of the best

heuristic solution is an indication of its effectiveness. Like DRA, the PTV and GTA heuristics produce only single sequences. PTV does so with an adaptation of Johnson's rule and GTA with a version of a slope index. It is to be expected that these single sequence procedures would not be as effective as others that choose from multiple sequences. That the average results are within seven and ten percent, respectively, of the best heuristic result is a testimony to the validity of the logic of their authors.

Interpretation of the factor main effects on the performance measure CE is also muddied by the presence of significant interaction. Detailed review of the n x m interaction effects gives similar but somewhat less pronounced results than was the case with SE. The n x h interaction shows different tendencies for different heuristics. At n=4, NEH produces an extremely poor CE while all others give excellent performance. As n increases, PTV, CDS, GTA, and DRA produce consistently low values of CE, RDM reflects gradually worsening performance, and NEH improves sharply at n=8 with continued slight improvement at higher levels. The three-way interaction shows consistently good performance for the four quick heuristics as above. NEH and RDM reflect similarly shaped results which vary consistently in magnitude. They reflect poorer performance for the smallest problem sizes at each level of n, i.e., 8x4, 12x4, 16x4, with NEH reflecting the greatest decline. Again, it would appear that some very general tendencies can be observed in the main factor effects despite the confounding effect of the significant interactions.

The differences in CE due to the main effects of n indicate that the computational efficiency for n=4 is significantly larger than for

any other level of this factor. This is due to the fact that the best heuristic solution for this level of n frequently occurs with one of the heuristics requiring the shortest processing time. Thus, the longer times of the RDM and NEH heuristics have a greater impact on average CE. At other levels of n, the best heuristic frequently occurs with one of the longer heuristics. When CE is calculated under these circumstances, the larger value of the divisor, T*, reduces the average value of this performance measure.

The differences is CE due to the main effects of the heuristics are much as expected. NEH, being the most complex heuristic, requires the longest processing time. Those occasions when it does not produce the best heuristic, or when another heuristic produces an identical best makespan, cause it to have a much larger average CE. Much the same thing can be said for the RDM heuristic wherein the processing time is strictly a function of the number of random sequences to be generated and tested. Although CDS has a slightly higher average CE, the other four heuristics do not differ significantly from each other. The processing times for these heuristics are all relatively short. When T* is produced by either NEH or RDM, as it frequently is, dividing the short processing times by a much larger T* produces an average value for CE of less than one.

4.1.3 Analysis of Achievement Measures

A summary of the achievement measures for each heuristic is given in Table XVI. These measures, taken together, provide an image of the lower end of the cumulative distribution of heuristic achievement as a percentage of best makespan results. It is obvious that NEH produces

TABLE XVI

SUMMARY OF ACHIEVEMENT MEASURES FOR PHASE ONE

HEUR	H_0	H_1	H_3	H_5
PTV	.05000 8 6	.08125 13 6	.24375 39 5	.40000 64 5
CDS	.23125 37 2	.33750 54 2	.60625 97 2	.78125 125 2
GTA	.10000 16 5	.11875 19 5	.19375 31 6	.29375 47 6
DRA	.11875 19 4	.15000 24 4	.26875 43 4	.49375 79 4
NEH	.90000 144 1	.92500 148 1	.96875 155 1	.98750 158 1
RDM	.21875 35 3	.33125 53 3	.53750 86 3	.74375 119 3

Each cell contains: Percentage
Number of Occurrences out of 160 problems
Relative Ranking of Heuristic

the best overall results. It produces the best makespan 90% of the time. Only twice in the 160 test problems did it fail to come within 5% of the best makespan. CDS is consistently in second place, barely edging out RDM which is consistently third. Although DRA is consistently fourth among the six heuristics, it fails to come within 5% of the best makespan more than 50% of the time. As was the case with the performance measure SE, GTA and PTV are far behind with GTA slightly outperforming PTV at H_0 and H_1 and reversing their positions at H_3 and H_5. This would indicate that GTA achieves the best makespan more often than PTV but when it misses, it tends to miss by a wider average margin.

It would appear that, on the basis of the achievement measures, NEH, CDS, and RDM are the only serious candidates for consideration as stand alone procedures. As Dannenbring [21] points out, however, these measures must be considered only in conjuction with the comparative performance measures. The achievement measures are consistent with the comparative measure SE, as well they should be since both are calculated from the same data elements. It is when one also considers CE that the true character of the heuristic comes to light. Comparing the three serious candidates, we find that NEH produces, by far, the best result but at considerable additional cost in computer processing time. Noting that, in many cases, more than one heuristic achieves the best makespan, we can see that CDS and RDM consistently produce good results in that 78% and 74%, respectively, are within 5% of the best solution. Both require considerably less processing time than does NEH with CDS requiring less than RDM.

With respect to a stand alone heuristic for flow shop scheduling,

it appears that management should choose between NEH with its associated high cost in terms of computer processing time and CDS which produces much quicker but somewhat less accurate results. Average computer processing times for these two heuristics for each problem size are given in Table XVII. The average times for NEH are more than 100 times those for CDS. The importance of speed versus accuracy must be weighed in each situation and the choice made as to which is the more important. However, the time differential that exists between the two heuristics would appear to be sufficient to warrant serious consideration of the faster but slightly less accurate CDS.

4.2 Phase Two

This phase of the research is concerned with the combination of heuristic methods as initializing procedures and the neighborhood search procedures for improving an initial solution. In addition to the six heuristics tested in phase one, an ordinal sequence of the jobs is also used to initialize the neighborhood search procedures, for a total of seven initialization procedures. These are combined with six neighborhood generating schemes, giving a total of 42 combinations. Two improvement rules are employed in the neighborhood search. Makespan was calculated for the sequence produced by each combination. Computer processing time in this phase includes the heuristic time to produce the initial solution as well the time required to generate and search the neighborhoods. As before, processing time is measured in milliseconds.

TABLE XVII

SUMMARY OF COMPUTER PROCESSING TIMES

FOR SELECTED HEURISTICS

Problem Size	Computer Processing Times (in milliseconds) CDS	NEH
4x4	1.0	412.9
4x8	2.0	622.9
4x12	4.0	849.9
4x16	7.0	1103.7
8x4	2.0	594.1
8x8	4.0	860.6
8x12	9.0	1143.0
8x16	14.3	1395.6
12x4	2.2	824.4
12x8	7.0	1037.0
12x12	14.0	1400.2
12x16	23.0	1639.6
16x4	3.0	946.7
16x8	10.0	1332.8
16x12	19.0	1553.5
16x16	32.0	1858.9
Average	9.59	1098.55

4.2.1 Analysis of Comparative Performance Measures

As in phase one, the best solution to each problem was identified and used to calculate values of SE and CE. These values were then subjected to an analysis of variance using the following models:

SE (or CE) = N M COMBO RULE N*M N*COMBO N*RULE M*COMBO
M*RULE RULE*COMBO N*COMBO*RULE M*COMBO*RULE
N*M*COMBO N*M*RULE N*M*COMBO*RULE

where COMBO = combination of initializing heuristic and neighborhood generating scheme

and RULE = improvement rule (first or best improvement)

The results of the analysis of variance on SE are given in Table XVIII. All of the main effects proved to be significant. Also significant were the two-way interactions N*M, N*COMBO, and M*COMBO, as well as the three-way interaction N*M*COMBO. The impact of these significant interactions is discussed in Section 4.2.2 below.

The REGWF multiple comparison procedure was applied to the main effects with the results given in Tables XIX through XXII. Table XIX shows that solution efficiency decreases (smaller is better) as the number of jobs increases, with no significant difference between 12 and 16 jobs. Table XX shows a similar trend for the number of machines with each level of this factor differing significantly from every other level. Table XXI shows that there are combinations or sets of combinations of initializing procedures and neighborhood generating schemes which differ significantly from other combinations or sets of combinations. Among the group of best combinations are all of those involving the removal and reinsertion of all pairs (IALP)

TABLE XVIII

ANOVA TABLE FOR PHASE TWO VARIABLE SE

DEPENDENT VARIABLE: SE

SOURCE	DF	SUM OF SQUARES	MEAN SQUARE
MODEL	1343	11.80412338	0.00878937
ERROR	12096	10.04526423	0.00083046
CORRECTED TOTAL	13439	21.84938760	

F VALUE	PR > F	R-SQUARE	C.V.
10.58	0.0	0.540250	2.8093
	ROOT MSE		SE MEAN
	0.02881773		1.02578317

SOURCE	DF	ANOVA SS	F VALUE	PR > F
N	3	1.64011281	658.31	0.0
M	3	0.28897488	115.99	0.0
COMBO	41	7.19664624	211.36	0.0
RULE	1	0.01310331	15.78	0.0001
N*M	9	0.68562461	91.73	0.0
N*COMBO	123	1.22484482	11.99	0.0
N*RULE	3	0.00108929	0.44	0.7264
M*COMBO	123	0.24223186	2.37	0.0001
M*RULE	3	0.00124767	0.50	0.6817
RULE*COMBO	41	0.01312375	0.39	0.9999
N*RULE*COMBO	123	0.02554158	0.25	1.0000
M*RULE*COMBO	123	0.01895781	0.19	1.0000
N*M*COMBO	369	0.40716329	1.33	0.0001
N*M*RULE	9	0.00043794	0.06	1.0000
N*M*RULE*COMBO	369	0.04499352	0.15	1.0000

TABLE XIX

MCP FOR MAIN EFFECTS OF N ON PHASE TWO VARIABLE SE

SAS

ANALYSIS OF VARIANCE PROCEDURE

RYAN-EINOT-GABRIEL-WELSCH MULTIPLE F TEST FOR VARIABLE: SE
NOTE: THIS TEST CONTROLS THE TYPE I EXPERIMENTWISE ERROR RATE

ALPHA=0.05 DF=12096 MSE=8.3E-04

NUMBER OF MEANS	2	3	4
CRITICAL F	5.00307	2.99647	2.60564

MEANS WITH THE SAME LETTER ARE NOT SIGNIFICANTLY DIFFERENT.

REGWF GROUPING	MEAN	N	N
A	1.0362674	3360	16
A			
A	1.0360905	3360	12
B	1.0206705	3360	8
C	1.0101043	3360	4

TABLE XX

MCP FOR MAIN EFFECTS OF M ON PHASE TWO VARIABLE SE

SAS

ANALYSIS OF VARIANCE PROCEDURE

RYAN-EINOT-GABRIEL-WELSCH MULTIPLE F TEST FOR VARIABLE: SE
NOTE: THIS TEST CONTROLS THE TYPE I EXPERIMENTWISE ERROR RATE

ALPHA=0.05 DF=12096 MSE=8.3E-04

NUMBER OF MEANS	2	3	4
CRITICAL F	5.00307	2.99647	2.60564

MEANS WITH THE SAME LETTER ARE NOT SIGNIFICANTLY DIFFERENT.

REGWF GROUPING	MEAN	N	M
A	1.0309103	3360	16
B	1.0288036	3360	12
C	1.0246593	3360	8
D	1.0187596	3360	4

TABLE XXI

MCP FOR MAIN EFFECTS OF COMBO ON PHASE TWO VARIABLE SE

RYAN-EINOT-GABRIEL-WELSCH MULTIPLE F TEST FOR VARIABLE: SE
NOTE: THIS TEST CONTROLS THE TYPE I EXPERIMENTWISE ERROR RATE

ALPHA=0.05 DF=12096 MSE=8.3E-04

NUMBER OF MEANS	2	3	4	5	6	7	8	9	10	11	12
CRITICAL F	9.18925	5.61369	4.29987	3.60505	3.17068	2.87134	2.65139	2.48228	2.34777	2.23794	2.14638
NUMBER OF MEANS	13	14	15	16	17	18	19	20	21	22	23
CRITICAL F	2.06874	2.00195	1.94384	1.8927	1.84735	1.8068	1.7703	1.73725	1.70717	1.67964	1.65435
NUMBER OF MEANS	24	25	26	27	28	29	30	31	32	33	34
CRITICAL F	1.63102	1.60942	1.58935	1.57066	1.55319	1.53683	1.52147	1.50701	1.49338	1.4805	1.46831
NUMBER OF MEANS	35	36	37	38	39	40	41	42			
CRITICAL F	1.45675	1.44577	1.43532	1.42537	1.41589	1.40683	1.39818	1.38987			

MEANS WITH THE SAME LETTER ARE NOT SIGNIFICANTLY DIFFERENT.

REGWF GROUPING	MEAN	N	COMBO
A	1.114711	320	39 (ORD-ADJP)
B	1.105099	320	37 (ORD-ADJD)
C	1.059215	320	13 (GTA-ADJD)
C	1.054823	320	1 (PTV-ADJD)
D	1.048274	320	14 (GTA-ADJP)
E	1.044292	320	19 (DRA-ADJD)
E	1.042915	320	31 (RDM-ADJD)
F	1.040230	320	2 (PTV-ADJP)
G	1.034055	320	7 (CDS-ADJD)
H	1.032526	320	32 (RDM-ADJP)
H I	1.030581	320	20 (DRA-ADJP)
H I J	1.029289	320	41 (ORD-IAJP)
H I J	1.028600	320	39 (ORD-ALLP)
I J	1.025117	320	8 (CDS-ADJP)
J	1.024922	320	17 (GTA-IAJP)
J K	1.023693	320	35 (RDM-IAJP)
K L	1.023674	320	5 (PTV-IAJP)

REGWF GROUPING	MEAN	N	COMBO
L M N O P	1.021795	320	15 (GTA-ALLP)
L M N O P	1.020336	320	23 (DRA-IAJP)
M N O P	1.019970	320	33 (RDM-ALLP)
R M N O P	1.019563	320	3 (PTV-ALLP)
R M N S O P	1.018630	320	11 (CDS-IAJP)
R T N S O P	1.017437	320	25 (NEH-ADJD)
R T U N S O P	1.016543	320	21 (DRA-ALLP)
R T U V S O P	1.016015	320	9 (CDS-ALLP)
R T U V S P	1.015788	320	26 (NEH-ADJP)
R T U V S W P	1.015574	320	40 (ORD-ISGL)
R T U V S W X	1.012881	320	34 (RDM-ISGL)
R T Y U V S W X	1.012740	320	16 (GTA-ISGL)
R T Y U V S W X	1.012667	320	29 (NEH-IAJP)
A T Z Y U V S W X	1.012211	320	4 (PTV-ISGL)
A T Z Y U V S W X	1.011791	320	27 (NEH-ALLP)
A T Z Y U V W X	1.010766	320	22 (DRA-ISGL)
A Z Y U V W X	1.010119	320	10 (CDS-ISGL)
A Z Y V W X	1.009047	320	28 (NEH-ISGL)
A Z Y W X	1.007610	320	42 (ORD-IALP)
A Z Y X	1.007386	320	36 (RDM-IALP)
A Z Y X	1.007059	320	18 (GTA-IALP)
A Z Y	1.006741	320	12 (CDS-IALP)
A Z	1.006316	320	6 (PTV-IALP)
A	1.005146	320	30 (NEH-IALP)
A	1.005744	320	24 (DRA-IALP)

TABLE XXII

MCP FOR MAIN EFFECTS OF RULE ON PHASE TWO VARIABLE SE

SAS

ANALYSIS OF VARIANCE PROCEDURE

RYAN-EINOT-GABRIEL-WELSCH MULTIPLE F TEST FOR VARIABLE: SE
NOTE: THIS TEST CONTROLS THE TYPE I EXPERIMENTWISE ERROR RATE

ALPHA=0.05 DF=12096 MSE=8.3E-04

NUMBER OF MEANS 2
CRITICAL F 3.84223

MEANS WITH THE SAME LETTER ARE NOT SIGNIFICANTLY DIFFERENT.

REGWF GROUPING	MEAN	N	RULE
A	1.0267706	6720	1
B	1.0247958	6720	2

which is the scheme that generates the largest neighborhoods. Among the worst combinations are those generated by exchanging adjacent doublets (ADJD) which produces the smallest neighborhoods. Table XXII shows that there is a significant difference due to the improvement rules and that best improvement (generating the entire neighborhood and taking the sequence with the greatest improvement) gives better solution efficiency.

The analysis of variance results for CE are given in Table XXIII. Here again, all of the main effects of the model are significant as are the interaction effects of N*M, N*COMBO, M*COMBO, M*RULE (at the 5% level), and N*M*COMBO. A summary of actions with respect to phase two hypotheses is given in Table XXIV.

The results of the multiple comparison procedure for the main effects are given in Tables XXV through XXVIII. Table XXV shows that computational efficiency increases (smaller is better) as the number of jobs increase with no significant difference for levels of 8 and 12 jobs. Table XXVI reflects a mixed effect of number of machines. The best CE occurs at m=8, followed by 16 and 12 with no significant difference. The worst case occurs at m=4. Table XXVII shows that there are combinations or sets of combinations which differ significantly for this performance measure from other combinations or sets thereof. Among the best performing combinations are those which combine the quickest initializing heuristics (PTV, GTA, and DRA) with the schemes which generate the smallest neighborhoods (ADJD and ADJP). At the other end of the performance scale are those combinations which pair the slowest heuristic (NEH) with any generating scheme and those which pair any heuristic with the scheme that generates the largest

TABLE XXIII

ANOVA TABLE FOR PHASE TWO VARIABLE CE

DEPENDENT VARIABLE: CE

SOURCE	DF	SUM OF SQUARES	MEAN SQUARE
MODEL	1343	181956745.79356695	135485.29098553
ERROR	12096	32863237.31451615	2716.86816423
CORRECTED TOTAL	13439	214819983.10808310	

F VALUE	PR > F	R-SQUARE	C.V.
49.87	0.0	0.847020	150.4485
	ROOT MSE		CE MEAN
	52.12358549		34.64545750

SOURCE	DF	ANOVA SS	F VALUE	PR > F
N	3	13047287.67396691	1600.78	0.0
M	3	1050297.07568368	128.86	0.0
COMBO	41	52672166.48101398	472.86	0.0
RULE	1	20652.87076568	7.60	0.0058
N*M	9	3775231.98940360	154.39	0.0
N*COMBO	123	97296932.54886331	291.16	0.0
N*RULE	3	5167.93920475	0.63	0.5930
M*COMBO	123	2488680.21206635	7.45	0.0
M*RULE	3	17397.81352075	2.13	0.0936
COMBO*RULE	41	94813.42835632	0.85	0.7373
N*COMBO*RULE	123	37302.98749110	0.11	1.0000
M*COMBO*RULE	123	104938.00125279	0.31	1.0000
N*M*COMBO	369	11241880.31594497	11.21	0.0
N*M*RULE	9	14125.04684132	0.58	0.8166
N*M*COMBO*RULE	369	89871.40869144	0.09	1.0000

TABLE XXIV

SUMMARY OF ACTIONS FOR PHASE TWO HYPOTHESES

HYPOTHESIS: There is no significant difference in SE (CE) due to:	SE	CE
N	Reject	Reject
M	Reject	Reject
COMBO	Reject	Reject
RULE	Reject	Reject
N*M	Reject	Reject
N*COMBO	Reject	Reject
N*RULE	Accept	Accept
M*COMBO	Reject	Reject
M*RULE	Accept	Reject
N*COMBO*RULE	Accept	Accept
M*COMBO*RULE	Accept	Accept
N*M*COMBO	Reject	Reject
N*M*RULE	Accept	Accept
N*M*COMBO*RULE	Accept	Accept

TABLE XXV

MCP FOR MAIN EFFECTS OF N ON PHASE TWO VARIABLE CE

SAS

ANALYSIS OF VARIANCE PROCEDURE

RYAN-EINOT-GABRIEL-WELSCH MULTIPLE F TEST FOR VARIABLE: CE
NOTE: THIS TEST CONTROLS THE TYPE I EXPERIMENTWISE ERROR RATE

ALPHA=0.05 DF=12096 MSE=2716.87

NUMBER OF MEANS	2	3	4
CRITICAL F	5.00307	2.99647	2.60564

MEANS WITH THE SAME LETTER ARE NOT SIGNIFICANTLY DIFFERENT.

REGWF GROUPING	MEAN	N	N
A	88.500	3360	4
B	19.313	3360	8
B	17.085	3360	12
C	13.684	3360	16

TABLE XXVI

MCP FOR MAIN EFFECTS OF M ON PHASE TWO VARIABLE CE

SAS

ANALYSIS OF VARIANCE PROCEDURE

RYAN-EINOT-GABRIEL-WELSCH MULTIPLE F TEST FOR VARIABLE: CE
NOTE: THIS TEST CONTROLS THE TYPE I EXPERIMENTWISE ERROR RATE

ALPHA=0.05 DF=12096 MSE=2716.87

NUMBER OF MEANS	2	3	4
CRITICAL F	5.00307	2.99647	2.60564

MEANS WITH THE SAME LETTER ARE NOT SIGNIFICANTLY DIFFERENT.

REGWF GROUPING	MEAN	N	M
A	49.369	3360	4
B	32.696	3360	12
B			
B	30.539	3360	16
C	25.978	3360	8

TABLE XXVII

MCP FOR MAIN EFFECTS OF COMBO ON PHASE TWO VARIABLE CE

RYAN-EINOT-GABRIEL-WELSCH MULTIPLE F TEST FOR VARIABLE: CE
NOTE: THIS TEST CONTROLS THE TYPE I EXPERIMENTWISE ERROR RATE

ALPHA=0.05 DF=12096 MSE=2716.87

NUMBER OF MEANS	2	3	4	5	6	7	8	9	10	11	12
CRITICAL F	9.18925	5.61369	4.29987	3.60505	3.17068	2.87134	2.65139	2.48228	2.34777	2.23794	2.14638
NUMBER OF MEANS	13	14	15	16	17	18	19	20	21	22	23
CRITICAL F	2.06874	2.00195	1.94382	1.8927	1.84735	1.8068	1.7703	1.73725	1.70717	1.67964	1.65435
NUMBER OF MEANS	24	25	26	27	28	29	30	31	32	33	34
CRITICAL F	1.63102	1.60942	1.58935	1.57066	1.55319	1.53683	1.52147	1.50701	1.49338	1.4805	1.46831
NUMBER OF MEANS	35	36	37	38	39	40	41	42			
CRITICAL F	1.45675	1.44577	1.43532	1.42537	1.41589	1.40683	1.39818	1.38987			

MEANS WITH THE SAME LETTER ARE NOT SIGNIFICANTLY DIFFERENT.

REGWF GROUPING	MEAN	N	COMBO
A	199.760	320	30 (NEH-IALP)
B	183.850	320	28 (NEH-ISGL)
B B	183.161	320	29 (NEH-IAJP)
B B	182.664	320	27 (NEH-ALLP)
B B	181.829	320	26 (NEH-ADJP)
B	181.757	320	25 (NEH-ADJD)
C C	41.833	320	36 (RDH-IALP)
D C C	40.039	320	42 (ORD-IALP)
D C	30.298	320	6 (PTV-IALP)
D	29.962	320	18 (GTA-IALP)
D	26.236	320	24 (DRA-IALP)
G F	24.005	320	12 (CDS-IALP)
H G F	19.816	320	34 (RDH-ISGL)
H G F	18.747	320	35 (RDH-IAJP)
H G F	18.171	320	33 (RDH-ALLP)
H G F	17.120	320	32 (RDH-ADJP)
H G	16.927	320	31 (RDH-ADJD)

REGWF GROUPING	MEAN	N	COMBO
H I	6.638	320	40 (ORD-ISGL)
I	4.469	320	41 (ORD-IAJP)
I	4.452	320	16 (GTA-ISGL)
I	4.326	320	4 (PTV-ISGL)
I	3.865	320	10 (CDS-ISGL)
I	3.829	320	22 (DRA-ISGL)
I	3.625	320	39 (ORD-ALLP)
I	3.103	320	17 (GTA-IAJP)
I	3.056	320	5 (PTV-IAJP)
I	2.832	320	23 (DRA-IAJP)
I	2.753	320	11 (CDS-IAJP)
I	2.402	320	15 (GTA-ALLP)
I	2.365	320	3 (PTV-ALLP)
I	2.279	320	9 (CDS-ALLP)
I	2.132	320	21 (DRA-ALLP)
I	1.073	320	8 (CDS-ADJP)
I	0.938	320	7 (CDS-ADJD)
I	0.881	320	33 (ORD-ADJP)
I	0.721	320	14 (GTA-ADJP)
I	0.639	320	2 (PTV-ADJP)
I	0.608	320	20 (DRA-ADJP)
I	0.547	320	37 (ORD-ADJD)
I	0.437	320	13 (GTA-ADJD)
I	0.423	320	1 (PTV-ADJD)
I	0.411	320	19 (DRA-ADJD)

TABLE XXVIII

MCP FOR MAIN EFFECTS OF RULE ON PHASE TWO VARIABLE CE

SAS

ANALYSIS OF VARIANCE PROCEDURE

RYAN-EINOT-GABRIEL-WELSCH MULTIPLE F TEST FOR VARIABLE: CE
NOTE: THIS TEST CONTROLS THE TYPE I EXPERIMENTWISE ERROR RATE

ALPHA=0.05 DF=12096 MSE=2716.87

NUMBER OF MEANS 2
CRITICAL F 3.84223

MEANS WITH THE SAME LETTER ARE NOT SIGNIFICANTLY DIFFERENT.

REGWF GROUPING	MEAN	N	RULE
A	35.8851	6720	2
B	33.4058	6720	1

neighborhood (IALP). Table XXVIII confirms that there is a significant difference in CE due to the improvement rule employed and that first improvement is more efficient than best improvement.

4.2.2 Interpretation of the Results

As was the case in phase one, the presence of significant interactions has a confounding effect on the interpretation of the impact of the main factors on SE. Detailed examination of the n x m interaction effect on SE reveals much the same situation as was described for phase one. Although there is some switching of positions at different levels of n and m, a general trend is still evident in plots of this interaction. The same is true for the n x combination interaction. With the exception of NEH combinations, all other combinations show a tendency to decreased performance as n increases. NEH combinations peak at n=12 and show a slight improvement of performance at n=16. The three-way interacton of n x m x combination reflects a similar pattern with slightly different magnitudes for the combinations across the spectrum of problem sizes. Again, it appears that, despite the obscurring effect of the interactons, some very general trends can be seen for the main factor effects.

The differences in SE due to the main effects of n and m can be interpreted in much the same way as was the case (in 4.1.2) for the heuristics as stand alone procedures.

The differences in SE due to the main effects of the combination of initializing procedures and neighborhood generating schemes are in keeping with intuitive logic. The largest neighborhood generated

(IALP) can be combined with any of the initializing procedures to produce very good results. With some exceptions, neighborhood size seems to be the primary determinant of solution efficiency. Any combination involving the most complex initializing procedure (NEH) seems to give reasonably good results. The worst performance of a combination involving NEH was ranked 20 out of 42 with an average SE of 1.01744 which means it missed the best makespan by less than two percent on average.

The differences in SE due to the main effects of improvement rules was very small but, nonetheless, significant. It would appear that, in terms of solution efficiency, it is better to take the best solution from each neighborhood as the starting point for the next search cycle.

A detailed review of the significant interaction effects on CE reveals much the same results as were observed for SE in this phase. Although the main effects are somewhat more obscurred by the interactions, there are still some fairly obvious general trends to be observed.

The differences in CE due to the main effects of n are attributed to much the same cause as was the case in phase one. The smaller divisor, T*, which occurs more frequently with smaller values of n, when combined with the longer processing times of some initializing procedures and neighborhood search routines, tend to inflate the values of CE.

The differences in CE due to the main effects of m are somewhat puzzling. The fact that level m=4 is the worst case can be attributed to much the same cause as that for factor n above. The puzzling

result is that level m=8 produces the best average result. One can only speculate that, at this level, the CE divisor, T*, is at its largest values because the best makespan occurs most frequently from those combinations that require the longest processing times. When the processing times of the shorter combinations are divided by this large divisor, the resulting CE is smallest on average. There does not appear to be any intuitive explanation for this phenomenon. Several additional problem sets would be required to determine if this is a general trend or merely an incidental occurence with the problem set generated for this study.

The differences in computational efficiency due to the main effects of the combination of initializing heuristics and neighborhood generating schemes are consistent with prior expectations. The primary determinant of CE, with the exception of those combinations involving NEH, is neighborhood size. Switching adjacent doublets (ADJD) and adjacent pairs (ADJP) provide the best results. Those combinations involving NEH provide, without exception, the worst case results because of the time required to produce the heuristic solution with which to initialize the neighborhood search procedure.

The differences in CE due to the main effects of the improvement rules is, for this measure also, small but significant. From the time efficiency standpoint, it is better to take the first improvement found in a neighborhood search as the starting sequence for the next search cycle. The first improvement rule provided a better average computer processing time in 72 of the 160 problems and a worse average processing time in only three cases. In most cases (136 Of 160) the resulting average makespan was the same for both improvement rules.

First improvement provided a better average makespan in only 11 cases and best improvement did better in only 13 cases. The three cases wherein first improvement required a greater processing time were all instances where it also provided a better solution. These results are discussed in greater detail in section 4.3.2.

4.2.3 Analysis of Achievement Measures

Summaries of the achievement measures for each combination of initializing heuristic and neighborhood generating scheme are given in Tables XXIX through XXXII. Table XXIX indicates the number of times each combination attains the best makespan. The combination of IALP with any heuristic gives good results with a slight edge to DRA as the initializing heuristic. The markedly better performance of NEH as an initializing heuristic for the schemes generating the smaller neighborhoods, ADJD and ADJP, is attributed to the number of times in the problems of larger size where NEH alone produced the best solution as the initial solution. The other heuristics did not produce a best solution initially as often and, therefore, did not perform as well in the subsequent search of the smaller neighborhoods. It is interesting to note that even the ordinal sequence, which often provided a relatively poor starting sequence, was performing on a par with the other initializing heuristics when combined with schemes that generated the largest neighborhoods, ISGL and IALP.

Table XXX shows the number of times each combination attained makespans within one percent of the best makespan. It is here that we can start to observe the situation mentioned in Dannenbring's [21] warning. Even when combined with the scheme generating the largest

TABLE XXIX

SUMMARY OF H_0 BY COMBO

Initial Heuristic	Search Routine ADJD	ADJP	ALLP	ISGL	IAJP	IALP
PTV	10	38	52	78	39	91
	12	39	65	82	38	103
	11.0	38.5	58.5	80.0	38.5	97.0
CDS	35	54	62	86	48	94
	35	53	65	87	44	103
	35.0	53.5	63.5	86.5	46.0	98.5
GTA	18	41	51	75	41	98
	17	42	61	83	41	92
	17.5	41.5	56.0	79.0	41.0	95.0
DRA	21	44	57	80	40	93
	18	43	67	81	46	107
	19.5	43.5	62.0	80.5	43.0	100.0
NEH	60	63	73	87	70	96
	61	64	78	88	71	96
	60.5	63.5	75.5	87.5	70.5	96.0
RDM	22	41	51	78	43	93
	22	41	54	79	45	97
	22.0	41.0	52.5	78.5	44.0	95.0
ORD	9	23	45	74	38	94
	10	27	53	80	45	102
	9.5	25.0	49.0	77.0	41.5	98.0

Cell Values: Number of times this combination attained best makespan:

Using first improvement rule
Using best improvement rule
Average achievement

TABLE XXX

SUMMARY OF H_1 BY COMBO

Initial Heuristic	Search Routine ADJD	ADJP	ALLP	ISGL	IAJP	IALP
PTV	14	50	70	101	57	126
	16	52	78	100	62	139
	15.0	51.0	74.0	100.5	59.5	132.5
CDS	44	64	79	104	68	125
	45	64	80	108	71	131
	44.5	64.0	79.5	106.0	69.5	128.0
GTA	24	50	67	92	57	124
	22	52	75	110	60	120
	23.0	51.0	71.0	101.0	58.5	122.0
DRA	24	53	74	96	57	125
	22	56	92	105	65	136
	23.0	54.5	83.0	100.5	61.0	130.5
NEH	78	83	97	113	96	131
	81	85	103	118	98	133
	79.5	84.0	100.0	115.5	97.0	132.0
RDM	30	52	68	95	57	125
	31	51	72	100	64	119
	30.5	51.5	70.0	97.5	60.5	122.0
ORD	12	26	52	90	49	123
	13	31	66	101	60	125
	12.5	28.5	59.0	95.5	54.5	124.0

Cell Values: Number of times this combination attained makespan within one percent of best makespan:

Using first improvement rule
Using best improvement rule
Average achievement

TABLE XXXI

SUMMARY OF H_3 BY COMBO

Initial Heuristic	Search Routine ADJD	ADJP	ALLP	ISGL	IAJP	IALP
PTV	39	72	109	137	108	153
	41	71	128	142	112	155
	40.0	71.5	118.5	139.5	110.0	154.0
CDS	81	100	130	143	122	148
	82	103	136	154	128	156
	81.5	101.5	133.0	148.5	125.0	152.0
GTA	42	68	109	131	100	153
	43	73	117	141	104	154
	42.5	70.5	113.0	136.0	102.0	153.5
DRA	52	87	121	142	146	160
	54	87	132	153	153	156
	53.0	87.0	126.5	147.5	149.5	158.0
NEH	124	128	140	147	137	155
	127	131	145	150	139	158
	125.5	129.5	142.5	148.5	138.0	156.5
RDM	66	88	122	140	101	158
	67	88	120	140	109	155
	66.5	88.0	121.0	140.0	105.0	156.5
ORD	18	32	87	125	83	152
	21	37	106	134	99	152
	19.5	34.5	96.5	129.5	91.0	152.0

Cell Values: Number of times this combination attained makespan within three percent of best makespan:

Using first improvement rule
Using best improvement rule
Average achievement

TABLE XXXII

SUMMARY OF H_5 BY COMBO

Initial Heuristic	Search Routine ADJD	ADJP	ALLP	ISGL	IAJP	IALP
PTV	77	102	148	157	141	160
	84	97	146	157	144	160
	80.5	99.5	147.0	157.0	142.5	160.0
CDS	118	138	154	160	152	159
	121	137	156	160	153	159
	119.5	137.5	155.0	160.0	152.5	159.0
GTA	73	91	139	152	134	160
	76	95	150	159	142	160
	74.5	93.0	144.5	155.5	138.0	160.0
DRA	98	121	155	159	146	160
	100	130	158	159	153	158
	99.0	125.5	156.5	159.5	149.5	159.0
NEH	147	149	157	158	154	158
	148	150	159	159	157	159
	147.5	149.5	158.0	158.5	155.5	158.5
RDM	101	117	147	159	139	160
	101	116	150	157	150	160
	101.0	116.5	148.5	158.0	144.5	160.0
ORD	34	44	123	146	128	160
	40	46	136	153	136	156
	37.0	45.0	129.5	149.5	132.0	158.0

Cell Values: Number of times this combination attained makespan within five percent of best makespan:

Using first improvement rule
Using best improvement rule
Average achievement

neighborhoods, several of the initializing heuristics (GTA, RDM, and ORD) are starting to fall behind the others indicating that, when they fail to produce the best makespan, they tend to miss by a wider margin. NEH still provides the best results over the smaller neighborhoods but is overtaken by PTV at the largest neighborhoods.

Table XXXI reflects the number of times each combination produces makespans within three percent of the best makespan. Again, we see several heuristics lagging at the largest neighborhood but the lag is not as marked as was the case at one percent. NEH, for the reasons previously discussed, is still best over the smaller neighborhoods and is overtaken, this time by DRA, at the largest neighborhood.

Table XXXII reflects production of makespans within five percent of the best solution. Here, we see further leveling of the performance of the initializing heuristics. Combined with the scheme generating the largest neighborhoods, all heuristics are capable of achieving solutions within five percent of the best makespan in virtually all problems.

Taking these results together, it would appear that the choice of initializing heuristic and neighborhood generating scheme depends upon management priorities with respect to accuracy (solution efficiency) and processing time to obtain the solution. If one is willing to accept a solution that is a little less accurate but that can be obtained quickly, then one can combine CDS or DRA with ISGL and be reasonably sure of obtaining a solution within five percent of the best solution in a comparatively short processing time. ISGL is chosen because of its much shorter processing time when compared to IALP. For example, for the CDS-ISGL combination, the average

processing time using the first improvement rule was 342.02 milliseconds, and was 464.23 milliseconds using the best improvement rule. In contrast, the respective average times using CDS-IALP were 3434.24 and 5309.81. Similarly, the average times for the DRA-ISGL combination were 369.81 and 495.50 compared to average times of 3781.99 and 6145.39 for the DRA-IALP combination. If, on the other hand, the major factor is accuracy, one is led to choose the DRA-IALP combination with a best improvement rule. This three-way combination will produce the best makespan better than 66% of the time.

4.3 Additional Analysis

4.3.1 Analysis of Neighborhood Size

Two of the research questions concerning neighborhood size require additional analysis. These are: (1) Does the neighborhood size account for the effectiveness of the search procedure?; and (2) Are there diminishing returns for larger neighborhoods?

Although previous analysis has given some indication that neighborhood size is a primary determinant of solution efficiency, one additional test of this preliminary indication was deemed appropriate. A correlation analysis of neighborhood size (NBH) and solution efficiency (FSE for the first improvement rule and BSE for the best improvement rule) for each value of n was performed. Since the value of n is the sole determinant of neighborhood size, it was felt that correlation analysis at each level of n would provide the best basis for comparison. The results of these analyses are given in Table XXXIII. The expected relationship is that SE will decrease as

neighborhood size increases. Thus, the correlation coefficients between NBH and either FSE or BSE should be negative in sign if the expected relationship holds. Table XXXIII shows that such is the case. However, the relationship is not as strong as might have been expected. The correlation coefficients are all significantly different from zero and range from approximately -.22 to -.33. It would appear that, although neighborhood size is a primary factor in accounting for the effectiveness of a search procedure, it is not the only factor that must be considered. We have already seen that there are significant differences due to the initializing heuristic and the number of machines.

In order to answer the question concerning diminishing returns for larger neighborhoods, a percentage of improvement achieved over the initial heuristic solution was calculated for each combination of initialization and search procedures for each problem. These were aggregated for each combination and the results appear in Table XXXIV. In general, Table XXXIV reflects a common pattern for all initializing heuristics. The first two incremental improvements show that the rate of improvement is increasing at an increasing rate. The one exception is with ORD where ADJP does not perform as well as ADJD. The incremental improvement peaks in all cases with the switch all pairs (ALLP) neighborhood generating scheme with a neighborhood size of $n(n-1)/2$. With the exception of the IAJP scheme which does not perform as well as ALLP, the remaining increases in neighborhood size reflect improvement at a decreasing rate. The absolute level of improvement achieved for each heuristic is generally, and inversely, related to the findings in phase one as to the goodness of the initial

TABLE XXXIII

CORRELATION ANALYSIS OF SELECTED VARIABLES

	NBH-FSE	NBH-BSE
N=4	-.25290 .0001	-.25297 .0001
N=8	-.28104 .0001	-.27997 .0001
N=12	-.22685 .0001	-.21997 .0001
N=16	-.33192 .0001	-.31473 .0001

Cell contents:

Pearson Correlation Coefficient
Probability >|R| under Ho: Rho=0

TABLE XXXIV

SUMMARY OF PERCENT IMPROVEMENT DATA BY HEURISTIC

	Neighborhood Size					
Initializing Heuristic	ADJD N-3	ADJP N-1	ALLP N(N-1)/2	IAJP (N-1)(N-2)	ISGL N(N-1)	IALP N(N-1)^2
PTV	.0291	.0425	.0610	.0571	.0675	.0728
	--	.0134	.0185	-.0039	.0104	.0053
CDS	.0119	.0202	.0286	.0261	.0342	.0373
	--	.0083	.0084	-.0025	.0081	.0031
GTA	.0486	.0589	.0821	.0792	.0899	.0970
	--	.0103	.0232	-.0029	.0170	.0071
DRA	.0281	.0408	.0534	.0499	.0587	.0633
	--	.0127	.0126	-.0035	.0088	.0046
NEH	.0018	.0034	.0073	.0065	.0099	.0127
	--	.0016	.0039	-.0008	.0034	.0028
RDM	.0079	.0176	.0293	.0257	.0359	.0410
	--	.0097	.0117	-.0036	.0102	.0051
ORD	.0776	.0712	.1400	.1393	.1506	.1570
	--	-.0064	.0688	-.0007	.0113	.0064
AVG	.0293	.0364	.0574	.0548	.0638	.0687
	--	.0071	.0210	-.0026	.0090	.0049

Cell contents:

Percent improvement over initial solution

Incremental improvement over previous neignborhood size
(as a percentage of initial solution)

solution provided by each heuristic.

4.3.2 Analysis of the Improvement Rules

The final research question to be answered concerns an analysis of the tradeoff between speed and accuracy for the two improvement rules. Close examination of the information contained in some of the previous tables should provide the answer to this question.

Table XVIII shows that there is a significant difference in solution efficiency (accuracy) due to the main effects of the improvement rules. Table XXII further indicates that the best improvement rule provides better solutions on average. Table XXIII indicates that there is a significant difference in computational efficiency (speed) due to the main effects of the improvement rules. Table XXVIII shows that the first improvement rule is faster on average. Tables XXIX through XXXII indicate the number of times each improvement rule attained the various levels of achievement. One can generally conclude from these tables that the best improvement rule does indeed provide better solutions. It should be noted, however, that the difference at the H_5 level is very slight.

To further assess the tradeoff between accuracy and speed in choosing an improvement rule, the results produced by each improvement rule were analized in detail. The first improvement rule produced a better makespan in 993 out of 6720 opportunities or 14.78% of the time. The best improvement rule produced a better makespan 1440 times or 21.43%. In 63.79% of the cases, the two rules produced identical makespans. In 65.07% of the cases where the best improvement rule produced a better makespan, the improvement was less than two percent.

The overall average was 1.94 percent and in only 14 cases did the amount of improvement exceed ten percent.

In a direct comparison of computational times, the first improvement rule required less time to reach a solution in 4158 cases (out of 6720) and the best improvement rule required less time in only 677 cases. The distributions of time differentials are shown in Table XXXV. If we disregard the differences of only one millisecond which could have resulted from the method of measurement, then the number of occurences favoring the first improvement rule reduces to 3645 while those favoring the best improvement rule reduce to 572. Not only are there far fewer instances favoring the best improvement rule, but three to five percent more of the differentials favoring the best improvement rule fall into the category of smaller differentials.

The first improvement rule will give an equal or better makespan approximately 79% of the time and will do so in much less time. The largest time differential favored the first improvement rule by more than 61,000 milliseconds. More than three percent of the time differentials favoring the first improvement rule did so by 8000 milliseconds or more.

So in choosing an improvement rule, managers are again faced with a choice between conflicting factors. If the primary determinant in the choice of scheduling techniques is accuracy, then the best improvement rule should be employed. If speed is of the essence and the manager is willing to accept a slight degradation in accuracy, the first improvement rule should be chosen. Each manger must decide the relative importance of speed and accuracy in his or her own situation, but it would appear that the additional accuracy provided by the best

TABLE XXXV

DISTRIBUTION OF COMPUTER PROCESSING TIME DIFFERENTIALS

Diff. Range Microsec.	First Improvement Rule Better No.	%	Cum. %	Best Improvement Rule Better No.	%	Cum. %
2-5	608	.1668	.1668	92	.1608	.1608
6-10	329	.0903	.2571	63	.1101	.2709
11-25	468	.1284	.3855	85	.1486	.4195
26-50	407	.1117	.4972	67	.1171	.5366
51-100	386	.1059	.6031	52	.0909	.6275
101-200	337	.0924	.6955	50	.0874	.7149
201-300	175	.0480	.7435	34	.0595	.7744
301-400	152	.0417	.7852	27	.0472	.8216
401-500	100	.0274	.8126	12	.0210	.8426
501-750	152	.0417	.8543	30	.0525	.8951
751-1000	91	.0250	.8793	15	.0262	.9213
1001-3000	199	.0546	.9339	21	.0367	.9580
3001-5000	76	.0208	.9542	14	.0245	.9825
5001-8000	52	.0143	.9690	4	.0070	.9895
8001-15000	56	.0154	.9844	4	.0070	.9965
15001-30000	43	.0118	.9962	2	.0035	1.0000
> 30000	14	.0038	1.0000	--		
Totals	3645			572		

improvement rule is not worth the added cost in computer processing time. This impression is reinforced when one compares the achievement measures of the two rules as reflected in Tables XXIX through XXXII.

CHAPTER V

SUMMARY AND CONCLUSIONS

5.1 Format

The summary and conclusions which follow will be couched in terms of the research questions which this study has attempted to answer. Each question is restated, followed by a summary of the findings and the conclusion drawn from them. Finally there is a discussion of some general conclusions and some recommendations concerning areas of further research in this area.

5.2 With Respect to Initialization Procedures

5.2.1 Which initialization procedure is best as a stand alone procedure and from what standpoint is it better?

The answer to this question must depend upon the primary concern of the manager with respect to the two factors of speed and accuracy. The heuristic proposed by Nawaz, Enscore, and Ham [52] consistently provides the best makespan results but at a cost of excessively long computer processing times. The Campbell, Dudek, and Smith [18] heuristic provides relatively good results at a much lower cost in processing time. The random heuristic also provides relatively good results but loses out in comparison to CDS because of its much longer

processing times.

One must conclude that NEH is the logical choice as a stand alone procedure for the manager who rates minimizing makespan (accuracy) above any consideration of the time required to produce the solution. For the manager that is willing to accept slight degradation in makespan in exchange for much quicker solutions, CDS is an excellent choice. CDS has the second rated solution efficiency, missing the best solution by less than three percent on average. For computational efficiency where smaller is better, CDS has an average value more than 100 times smaller than that for NEH.

5.2.2 Does the choice of the initialization procedure depend upon the search procedure to be subsequently employed?

The information to answer this question can be derived from Table XXI. The ratings of the heuristics as an initialization procedure for each of the neighborhood generating schemes have been extracted and are shown in Table XXXVI. This table shows that there are slight differences in the ratings of the heuristics among the six neighborhood generating schemes. The two heuristics that give the two best solution efficiencies are identical over all generating schemes except IALP. NEH gives the best results with each scheme and CDS provides the second best results. There are some differences at the lower rankings among the schemes. In the case of IALP, it appears that the larger neighborhoods generated cause a change in the relative ranking of the heuristics as initializers. DRA takes over the top spot while NEH drops to second. CDS drops to fourth place following

TABLE XXXVI

RANKINGS OF HEURISTICS AS INITIALIZATION PROCEDURES

Rank	ADJD		ADJP		ALLP		IAJP		ISGL		IALP	
	First	Best	First	Best	First	Best	First	Best	First	Best	First	Best
1	NEH	NEH	NEH	NEH	NEH	NEH	NEH	NEH	NEH	NEH	DRA	DRA
2	CDS	CDS	CDS	CDS	CDS	DRA	CDS	CDS	CDS	CDS	NEH	PTV
3	RDM	RDM	DRA	DRA	DRA	CDS	PTV	DRA	DRA	DRA	GTA	NEH
4	DRA	DRA	RDM	RDM	RDM	PTV	RDM	RDM	PTV	GTA	RDM	CDS
5	PTV	PTV	PTV	PTV	PTV	GTA	DRA	PTV	RDM	PTV	PTV	GTA
6	GTA	GTA	GTA	GTA	GTA	RDM	GTA	GTA	GTA	RDM	CDS	ORD
7	ORD	ORD	ORD	ORD	ORD	ORD	ORD	ORD	ORD	ORD	ORD	RDM

PTV.

When one considers the time factors from Table XXVII as well as the solution efficiencies discussed above, it appears that CDS is the best choice as an initializing heuristic for all neighborhood generating schemes except IALP for which DRA is the appropriate choice. One must also recall, however, that IALP also generates the largest neighborhood and takes the longest processing time. If a manager is willing to accept slight degradation in solution efficiency to get much faster solutions, then we must again conclude that CDS in combination with ISGL will give consistently good results in a limited amount of time.

We can conclude that the choice of an initialization procedure does depend somewhat upon the search procedure (or more specifically, the neighborhood generating scheme) to be subsequently employed. DRA should be used to initialize IALP. If one is not concerned with time, NEH should initialize all other generating schemes. CDS should initialize the other schemes when time is also considered.

5.3 With Respect to Neighborhood Search Procedures

5.3.1 Does the neighborhood size account for the effectivenes of the search procedure?

The correlation coefficients in Table XXXIII indicate that there is correlation between neighborhood size and solution efficiency. However, the stength of the relationship (-.22 to -.33) is not as strong as might have been expected. Table XXVII also indicates that neighborhood size is a major factor in search routine performance.

One can conclude that neighborhood size is a key factor in determining the effectiveness of a search routine. It is not, however, the only factor that determines effectiveness. Table XXXIV indicates that IAJP, although it generates a larger neighborhood than ALLP, consistently produces poorer results. ALLP generates the largest neighborhood of the schemes which are created by exchanging (or switching) pairs. IAJP, on the other hand, generates the smallest neighborhood of the schemes which remove and reinsert one or more jobs in the sequence. It appears, therefore, that the pattern or method of neighborhood generation also plays a significant role in determining the effectiveness of a search routine.

5.3.2 Are there diminishing returns for larger neighborhoods?

With some minor exceptions and one glaring one, increasing neighborhood size does produce diminishing returns. This can be seen in the incremental improvements reflected in Table XXXIV. The glaring exception is search routine IAJP which creates a larger neighborhood than ALLP but produces poorer solutions. The neighborhood generation schemes used in this study can be divided into two general approaches: exchanging pairs of jobs and removal/reinsertion of one or more jobs within a sequence. Within each of these approaches, there is a distinct pattern of diminishing returns.

We can conclude that, in general, increasing neighborhood size does produce diminishing returns. This becomes particularly evident when the time factor is also considered. IALP which produces the best levels of solution efficiency also produces the poorest levels of

computational efficiency. A manager willing to accept slightly reduced solution efficiency can obtain consistently good solutions with greatly reduced computational times by employing a neighborhood generation scheme such as ISGL.

5.3.3 What tradeoffs, in terms of computational speed versus solution effectiveness, are involved in using a first improvement rule rather than a best improvement rule?

As is usually the case, better solutions require more time to achieve. Such is the case here. The best improvement rule provides better solutions in slightly more than 21% of the cases compared to just under 15% for the first improvement rule. The first improvement rule acieved a solution in less time in approximately 62% of the cases compared to approximately 10% for the best improvement rule.

The question that managers must resolve is whether the improved solution is worth the additional computational time. This question must be answered in light of the unique circumstances existing in each organization. However, in most organizations, it would appear that the slight improvement in solutions obtained by using the best improvement rule would not justify the additional time required to obtain them.

5.4 General Conclusions

The increasing use of group technology and cellular manufacturing provides an opportunity for practical application of flow shop scheduling heuristics to a degree that has not previously existed.

This study has attempted to provide new insights into the intricacy of flow shop scheduling heuristics and neighborhood search routines.

The Nawaz, Enscore, and Ham heuristic provides the best performance, in terms of solution efficiency, of the heuristics tested as stand alone procedures. It also requires an excessive amount of computational time compared to other heuristics. The manager is forced to choose between the best solution requiring excessive time and a slightly degraded solution produced by another heuristic, such as Campbell, Dudek, and Smith, in a much shorter time. It should be noted that the testing of stand alone procedures in phase one of this study is somewhat biased in that it tested only selected heuristics that were to be used in phase two as initialization procedures for the neighborhood search routines. Other heuristics, such as Dannenbring's extensive search procedure which in itself employs a neighborhood search routine, were not tested in phase one because they were part of the phase two tests. Previous research (see Turner and Booth [73]) has shown that NEH still provides greater solution efficiency but did not make similar comparisons for computational efficiency. It appears that additional study in this area may be warranted.

Improvements can be made to the stand alone heuristics by the addition of a systematic neighborhood search routine. The goodness of the resulting solution depends, in part, on the size of the neighborhood generated and the improvement rule employed. Heuristic initialization procedures give better results than can be obtained with an arbitrary starting sequence such as an ordinal sequence. The best results, in terms of solution efficiency, were obtained by combining a fairly quick heuristic sequence, DRA, with the scheme

generating the largest neighborhoods, IALP. This combination, because of the time required to search the largest neighborhoods, also required excessive computational time. The combination of a quick heuristic, CDS, with a generation scheme generating smaller neighborhoods, ALLP or ISGL, appears to offer a reasonable balance between speed and accuracy.

5.5 Directions for Future Research

Several areas requiring further research have come to light during the course of this study. The results of this study are based upon job processing times randomly generated from a uniform (0-99) distribution. It was noted that in cellular manufacturing, which offers the most promising application, processing times on the machines for a single job are likely to show a high degree of correlation because the cell will have been designed that way with multiple machines for the slower operations. The primary variance will occur due to lot sizes of the jobs awaiting processing. This research should be replicated with a problem set reflecting correlated job times to determine if the findings herein will hold for such situations.

It was also noted that there is a pattern of diminishing returns for increasing neighborhood sizes, particularly within a given approach to neighborhood generation. One must wonder if better solutions could be obtained, and at what cost in computer processing time, were one to use a mixture of neighborhood generation strategies either alternately or sequentially. The software programs written to support the current research can be readily modified to support

further research into neighborhood generation schemes.

Finally, the phase one tests of stand alone heuristics did not include those which combine an initiation procedure with a neighborhood search procedure, such as Dannenbring's extensive search or the Turner modification to NEH. Although these procedures were tested as part of the phase two tests, an extensive comparison of the stand alone heuristics and the combined procedures still needs to be made to make the phase one findings more complete.

SELECTED BIBLIOGRAPHY

[1] Amar, A. D., and J. N. D. Gupta, "Simulated Versus Real Life Data in Testing the Efficiency of Scheduling Algorithms", IIE Transactions, 18 (1986), pp. 16-25.

[2] APICS Dictionary, Fifth Edition, (Thomas F. Wallace, editor), American Production and Inventory Control Society, Inc., Falls Church, Va., 1984.

[3] Ashour, Said, "A Decomposition Approach for the Machine Scheduling Problem", International Journal of Production Research, 6 (1968), pp. 109-122.

[4] _____ , "A Modified Decomposition Algorithm for Scheduling Problems", International Journal of Production Research, 8 (1970), pp. 281-284.

[5] Axsater, Sven, "On Scheduling in a Semi-ordered Flow Shop Without Intermediate Queues", IIE Transactions, 14 (1982), pp. 128-130.

[6] Baker, Kenneth R., Introduction to Sequencing and Scheduling, John Wiley and Sons, New York, 1974.

[7] _____ , "A Comparative Study of Flow-shop Algorithms", Operations Research, 23 (1975), pp. 62-73.

[8] _____ , "An Elimination Method for the Flow Shop Problem", Operations Research, 23 (1975), pp. 159-162.

[9] Bakshi, M. S., and S. R. Arora, "The Sequencing Problem", Management Science, 16 (1969), pp. B247-B263.

[10] Balas, E., "Machine Sequencing Via Disjuctive Graphs: An Implicit Enumeration Algorithm", Operations Research, 17 (1969), pp. 941-957.

[11] Bansal, S. P., "Minimizing the Sum of Completion Times of n Jobs Over m Machines in a Flowshop", IIE Transactions, 9 (1977), pp. 306-311.

[12] Bestwick, P. F., and N. A. J. Hastings, "A New Bound for Machine Scheduling", Operational Research Quarterly, 27 (1976), pp. 479-487.

[13] Bonney, M. C., and S. W. Gundry, "Solutions to the Constrained Flowshop Sequencing Problem", Operational Research Quarterly, 27 (1976), pp. 869-883.

[14] Bowman, E. H., "The Schedule-Sequencing Problem", Operations Research, 7 (1959), pp. 621-624.

[15] Brooks, G. H., and C. R. White, "An Algorithm for Finding Optimal or Near Optimal Solutions to the Production Scheduling Problem", Journal of Industrial Engineering, 16 (1965), pp. 34-40.

[16] Bruvald, N. T., and J. R. Evans, "Flexible Mixed-Integer Programming Formulations for Production Scheduling", IIE Transactions, 17 (1985), pp. 2-7.

[17] Burns, Fennell, and John Rooker, "Three Stage Flow-shops with Recessive Second Stage", Operations Research, 26 (1978), pp. 207-208.

[18] Campbell, H. G., R. A. Dudek, and M. L. Smith, "A Heuristic Algorithm for the n Job m Machine Sequencing Problem", Management Science, 16 (1970), pp. B630-B637.

[19] Conover, W. J., Practical Nonparametric Statistics, 2d edition, John Wiley and Sons, New York, 1980.

[20] Conway, R. W., W. L. Maxwell, and L. W. Miller, Theory of Scheduling, Addison-Wesley, Reading, Mass., 1967.

[21] Dannenbring, David G., "The Evaluation of Heuristic Solution Procedures for Large Combinatorial Problems", (unpublished Ph. D. dissertation, Columbia University, New York, 1973).

[22] _____ , "An Evaluation of Flow Shop Sequencing Heuristics", Management Science, 23 (1977), pp. 1174-1182.

[23] Dantzig, G. B., "Discrete Variable Extremum Problems", Operations Research, 5 (1957), pp. 266-276.

[24] Donaldson, T. S., "Power of the F-Test for Nonnormal Distributions and Unequal Variances", Report no. RM-5072-PR, Rand Corporation, Santa Monica, Calif., 1966.

[25] Dudek, R. A., and O. F. Teuton, Jr., "Development of M-Stage Decision Rule for Scheduling n Jobs Through m Machines", Operations Research, 12 (1965), pp. 471-497.

[26] Dunnet, C. W., "A Multiple Comparison Procedure for Comparing Several Treatments with a Control', Journal of the American Statistics Association, 50 (1955), pp. 1086-1121.

[27] Einot, I., and K. R. Gabriel, "A Study of the Powers of Several Methods of Multiple Comparisons", Journal of the American Statistical Association, 70 (1975), pp. 574-583.

[28] French, S., Sequencing and Scheduling: An Introduction to the Mathematics of the Job Shop, Ellis Horwood Ltd, Chichester, England, 1982.

[29] Gabriel, K. R., "A Simple Method of Multiple Comparison of Means",/Journal of the American Statistical Association/73 (1978), pp. 724-729.

[30] Garey, M. R., and D. S. Johnson, Computers and Intractability - A Guide to NP-Completeness, W. H. Freeman and Co., San Francisco, 1979.

[31] Giffler, B., and G. L. Thompson, "Algorithm for Solving Production Scheduling Problems", Operations Research, 8 (1960), pp. 487-503.

[32] Giglio, R. J., and H. M. Wagner, "Approximate Solutions to the Three-Machine Scheduling Problem", Operations Research, 12 (1964), pp. 305-324.

[33] Gomory, R. E., "Outline of an Algorithm for Integer Solutions to Linear Problems", Bulletin of the American Mathematical Society, 64 (1958), pp. 275-278.

[34] _____ , "An All-Integer Programming Algorithm", Industrial Scheduling, Chapter 13, (J. F. Muth and G. L. Thompson, editors), Prentice-Hall, Inc., Englewood Cliffs, N. J., 1963.

[35] Graves, S. C., "A Review of Production Scheduling", Operations Research, 29 (1981), pp. 646-675.

[36] _____ , H. C. Meal, D. Stefek, and A. H. Zeghmi, "Scheduling of Reentrant Flow Shops", Journal of Operations Management, 3 (1983), pp. 197-207.

[37] Gupta, J. N. D., "A Functional Heuristic Algorithm for the Flowshop Scheduling Problem", Operational Research Quarterly, 22 (1971), pp. 39-47.

[38] _____, and R. A. Dudek, "Optimality Criteria for Flow Shop Schedules", AIIE Transactions, 3 (1971), pp. 199-205.

[39] Heller, J., "Some Numerical Experiments for an M x J Flow Shop and its Decision-Theoretic Aspects", Operations Research, 8 (1960), pp. 178-184.

[40] Ignall, E., and L. Schrage, "Application of the Branch and Bound Technique to Some Flow-Shop Scheduling Problems", Operations Research, 13 (1965), pp. 400-412.

[41] Johnson, S. M., "Optimal Two- and Three-Stage Production Schedules With Setup Times Included", Naval Research Logistics Quarterly, 1 (1954), pp. 61-68.

[42] King, J. R., and A. S. Spachis, "Heuristics For Flow Shop Scheduling", International Journal of Production Research, 18 (1980), pp. 345-357.

[43] Kleijnen, J. P. C., Statistical Techniques in Simulation, Part II, Marcel Dekker, Inc., New York, 1975.

[44] Krone, M. J., and K. Steiglitz, "Heuristic Programming Solution of a Flowshop Scheduling Problem", Operations Research, 22 (1974), pp. 629-638.

[45] Lagaweg, B. J., J. K. Lenstra, and A. H. G. Rinooy Kan, "A General Bounding Scheme for the Permutation Flow Shop Problem", Operations Research, 26 (1978), pp. 53-67.

[46] Little, J. D. C., K. G. Murty, D. M. Sweeny, and C. Karel, "An Algorithm for the Traveling Salesman Problem", Operations Research, 11 (1963), pp. 972-989.

[47] Lomnicki, Z., "A Branch-and-Bound Algorithm for the Exact Solution of the Three-Machine Scheduling Problem", Operational Research Quarterly, 16 (1965), pp. 89-100.

[48] Manne, A. S., "On the Job Shop Scheduling Problem", Operations Research, 8 (1960), pp. 219-223.

[49] Markowitz, H. M., and A. S. Manne, "On the Solution of Discrete Programming Problems", Econometrica, 25 (1957), pp. 84-110.

[50] McMahon, G. B., and P. G. Burton, "Flow-Shop Scheduling With the Branch and Bound Method", Operations Research, 15 (1968), pp. 473-481.

[51] Muth, J. F., "The Effects of Uncertainty in Job Times on Optimal Schedules', Industrial Scheduling, Chapter 18, (J. F. Muth and G. L. Thompson, editors), Prentice-Hall, Inc., Englewood Cliffs, N. J., 1963.

[52] Nawaz, M., E. E. Enscore, Jr., and I. Ham, "A Heuristic Algorithm for the m-Machine n-Job Flow-Shop Sequencing Problem", Omega, 11 (1983), pp. 91-95.

[53] Page, E. S., "An Approach to Scheduling Jobs on Machines", Journal of the Royal Statistical Society, Series B, 23 (1961), pp. 484-492.

[54] Palmer, D. S., "Sequencing Jobs Through a Multi-stage Process in the Minimum Total Time - A Quick Method of Obtaining a Near Optimum", Operational Research Quarterly, 16 (1965), pp. 101-107.

[55] Panwalkar, S. S., R. A. Dudek, and L. M. Smith, "Sequencing Research and the Industrial Scheduling Problem", Symposium on the Theory of Scheduling and its Application, (S. E. Elmaghraby, editor), Springer-Verlag, New York, 1973, pp. 29-38.

[56] _____ , and A. W. Kahn, "An Ordered Flow-shop Sequencing Problem With Mean Completion Time Criteria", International Journal of Production Research, 14 (1976), pp. 631-635.

[57] Park, Y. B., "A Simulation Study and an Analysis of Performance Effectiveness of Flowshop Sequencing Heuristics: A Static and a Dynamic Model", unpublished Master's Thesis, Pennsylvania State University, 1981).

[58] Petrov, V. A., Flowline Group Production Planning, (E. Bishop, translator), Business Publications, Ltd., London, 1968.

[59] Pounds, W. F., "The Scheduling Environment", Industrial Scheduling, Chapter 1, (J. F. Muth and G. L. Thompson, editors), Prentice-Hall, Inc., Englewood Cliffs, N. J., 1963.

[60] Rinnooy Kan, A. H. G., Machine Scheduling Problems, Martinus Nijhoff, The Hague, 1976.

[61] Ryan, T. A., "Significance Tests for Multiple Comparisons of Proportions, Variances, and Other Statistics", Psychological Bulletin, 57 (1960), pp. 318-328.

[62] Saaty, Thomas L., The Analytic Hierarchy Process, Mc Graw-Hill, New York, 1980.

[63] SAS User's Guide: Statistics (1982), SAS Institute Inc., Carey, N. C.

[64] Setiaputra, W., "A Survey of Flow-shop Permutation Scheduling Techniques and an Evaluation of Heuristic Solution Methods", (unpublished Master's Thesis, Pennsylvania State University, 1980).

[65] Scheffe, H., The Analysis of Variance, Wiley, New York, 4th Printing, 1964.

[66] Sisson, R. L., "Sequencing Theory", Progress in Operations Research, Vol. 1, Chapter 7, (R. L. Ackoff, editor), John Wiley and Sons, New York, 1961.

[67] Skinner, W., "The Focused Factory", Harvard Business Review, May-Jun, 1974, pp. 40-48.

[68] Story, A. E., and H. M. Wagner, "Computational Experience With Integer Programming for Job Shop Scheduling", Industrial Scheduling, Chapter 14, (J. F. Muth and G. L. Thompson, editors), Prentice-Hall, Inc., Englewood Cliffs, N. J., 1963.

[69] Szwarc, W., "Elimination Methods in the m x n Sequencing Problem", Naval research Logistics Quarterly, 18 (1971), pp. 295-305.

[70] _____ , "The Flow Shop Problem With Mean Completion Time Criteria", IIE Transactions, 15 (1983), pp. 172-176.

[71] Turner, Scott, "Modifications of Dannenbring's and Nawaz, Enscore, and Ham's n Job m Machine Heuristics for Improved Performance", (unpublished research report, Oklahoma State University, 1985).

[72] _____ , and Dean Booth, "A New Integer Programming Model for the N Job M Machine Flow Shop Problem", Proceedings, 17th Annual Meeting of the Midwest Decision Sciences Institute, Lincoln, Neb., Apr. 23-25, 1986.

[73] _____ , "Comparison of Heuristics for Flowshop Sequencing", Omega, 15 (1987), pp. 75-78.

[74] Wagner, H. M., "An Integer Linear Programming Model for Machine Scheduling", Naval Research Logistics Quarterly, 6 (1959), pp. 131-140.

[75] Welsch, R. E., "Stepwise Multiple Comparison Procedures", Journal of the American Statistical Association 72 (1977), pp. 566-578.

[76] Winer, B. J., Statistical Principles in Experimental Design, McGraw-Hill, New York, 1971.

APPENDIX

LISTING OF COMPUTER PROGRAMS

```
C      PROCESSING TIME GENERATOR
C
C      THIS PROGRAM GENERATES THE U(0,99) RANDOM PROCESSING
C      TIMES FOR THE PROBLEM SET WITH UNCORRELATED PROCESSING
C      TIMES.  OUTPUT IS PLACED IN A FILE FROM WHICH IT CAN
C      BE READ FOR SUBSEQUENT PROBLEM SOLUTION.
C
C      THIS PROGRAM CONSISTS OF A DRIVER ROUTINE WHICH SETS
C      PROBLEM SIZE AND THE SEED FOR RANDOM NUMBER GENERATION,
C      AND A SUBROUTINE WHICH GENERATES THE RANDOM PROCESSING
C      TIMES AND PUTS THEM INTO THE OUTPUT FILE.
C
       INTEGER SEED, T(16,16)
C
     5 READ(5,10,END=100) N,M,SEED
C
       WRITE(8,15) N,M,SEED
C
       CALL RGEN(N,M,SEED)
       GOTO 5
C
    10 FORMAT(2I3,I7)
    15 FORMAT(' ',2I3,I7)
C
   100 STOP
       END
C
C
       SUBROUTINE RGEN(N,M,SEED)
C
       INTEGER SEED, T(16,16), H
C
       RNDM = RANF(SEED)
C
       DO 40 K=1,10
          DO 30 I=1,N
             DO 20 J=1,M
                T(I,J) = (RANF(0)*100)
    20       CONTINUE
          WRITE(8,50) (T(I,H),H=1,M)
    30    CONTINUE
       WRITE(8,60)
    40 CONTINUE
C
    50 FORMAT(' ',16I3)
    60 FORMAT(1X)
C
       RETURN
       END
```

```
 4  4 691272
 4  8 642801
 4 12 890146
 4 16 278387
 8  4 349111
 8  8 920575
 8 12 754720
 8 16 934617
12  4 471524
12  8 862776
12 12 266438
12 16 560064
16  4 163215
16  8 461950
16 12 184411
16 16 231908
//
```

```
C
C     HUERISTIC APPLICATION PROGRAM
C
C     THIS PROGRAM READS THE FILE CONTAINING THE RANDOMLY
C     GENERATED PROBLEMS, APPLIES EACH OF THE SIX HUERISTICS
C     IN TURN, AND OUTPUTS THE SEQUENCE, MAKESPAN, AND
C     COMPUTER PROCESSING TIME FOR EACH HUERISTIC ON EACH
C     PROBLEM IN THE PROBLEM SET.
C
      INTEGER T(16,16),S(16),X,TIME
      CHARACTER MNE*3
C
C     READ NUMBER OF JOBS AND NUMBER OF MACHINES FROM THE
C     DATA SET.
C
    5 READ(8,30,END=100) N,M,X
      WRITE(9,40) N,M
C
C     OUTER LOOP FOR EACH PROBLEM OF A GIVEN SIZE.
C
      DO 20 K=1,10
C
C     READ IN PROCESSING TIME MATRIX.
C
      DO 10 I=1,N
      READ(8,50) (T(I,J),J=1,M)
   10 CONTINUE
C
      MNE='PTV'
      CALL PTV(M,N,T,TIME,S,ICPT)
      WRITE(9,60) MNE,K,TIME,ICPT,(S(I),I=1,N)
C
      MNE='CDS'
      CALL CDS(M,N,T,MIN,S,ICPT)
      WRITE(9,60) MNE,K,MIN,ICPT,(S(I),I=1,N)
C
      MNE='GTA'
      CALL GTA(M,N,T,TIME,S,ICPT)
      WRITE(9,60) MNE,K,TIME,ICPT,(S(I),I=1,N)
C
      MNE='DRA'
      CALL DRA(M,N,T,MIN,S,ICPT)
      WRITE(9,60) MNE,K,MIN,ICPT,(S(I),I=1,N)
C
      MNE='NEH'
      CALL NEH(M,N,T,MIN,S,ICPT)
      WRITE(9,60) MNE,K,MIN,ICPT,(S(I),I=1,N)
C
      MNE='RDM'
      CALL RDM(M,N,T,MIN,S,ICPT)
      WRITE(9,60) MNE,K,MIN,ICPT,(S(I),I=1,N)
      WRITE(9,80)
C
      READ(8,70)
```

```
C
   20 CONTINUE
C
      GOTO 5
C
   30 FORMAT(I4,I3,I7)
   40 FORMAT(' ',2I3)
   50 FORMAT(I4,15I3)
   60 FORMAT(' ',A4,I3,I5,I5,16I2)
   70 FORMAT(1X)
   80 FORMAT(1X)
C
  100 STOP
      END
C
C
      SUBROUTINE PTV(M,N,T,TIME,S,ICPT)
C
      INTEGER S(16),T(16,16),TIME,A(16),B(16),H
C
      CALL ELAPSE(ICPT)
C
C     DETERMINE INDEX FOR ONE-HALF OF MACHINE SET.
C     IF M IS ODD, CENTER MACHINE INCLUDED IN BOTH HALVES.
C
      H=(M+1)/2
C
C     CLEAR ARRAYS A AND B.
C
      DO 700 I=1,N
         A(I)=0
         B(I)=0
  700 CONTINUE
C
C     SUM TIMES OF FIRST HALF OF MACHINES INTO ARRAY A
C     AND SECOND HALF INTO ARRAY B.
C
      DO 730 I=1,N
         DO 710 J=1,H
  710       A(I)=A(I)+T(I,J)
         DO 720 J=M-H+1,M
  720       B(I)=B(I)+T(I,J)
  730 CONTINUE
C
C     APPLY JOHNSON'S RULE TO FIRST AND SECOND HALF SUMS.
C
      CALL JOHN(A,B,S,N)
C
C     CALCULATE MAKESPAN FOR SINGLE RESULTING SEQUENCE.
C
      CALL MKSP(M,N,T,S,TIME)
C
      CALL ELAPSE(ICPT)
C
      RETURN
      END
C
C
C
      SUBROUTINE CDS(M,N,T,MIN,S,ICPT)
```

CDS

```
C
      INTEGER A(16),B(16),S(16),T(16,16),SA(16),TIME
C
      CALL ELAPSE(ICPT)
C
      MIN=100000
C
C     OUTER LOOP CREATES M-1 TWO MACHINE SUBPROBLEMS
C
      DO 495 K=1,M-1
C
C     CLEAR ARRAYS A AND B TO PROCESS CURRENT SUBPROBLEM
C
      DO 450 I=1,N
         A(I)=0
         B(I)=0
  450 CONTINUE
C
C     COMPUTE SUBPROBLEM PROCESSING TIMES AND PUT IN ARRAYS
C     A AND B.
C
      DO 470 I=1,N
         DO 460 L=1,K
            A(I)=A(I)+T(I,L)
            B(I)=B(I)+T(I,M-L+1)
  460    CONTINUE
  470 CONTINUE
C
C     APPLY JOHNSON'S RULE TO EACH SUBPROBLEM.
C
      CALL JOHN(A,B,S,N)
C
C     COMPUTE MAKESPAN FOR CURRENT SEQUENCE AND COMPARE TO
C     CURRENT MINIMUM.  UPDATE MINIMUM TIME AND SEQUENCE
C     ARRAY AS NECESSARY.
C
      CALL MKSP(M,N,T,S,TIME)
      IF (TIME .LT. MIN) THEN
      DO 480 II=1,N
         SA(II)=S(II)
  480 CONTINUE
      MIN=TIME
      ENDIF
  495 CONTINUE
C
C     REVISE S TO REFLECT BEST SEQUENCE.
C
      DO 490 I=1,N
         S(I)=SA(I)
  490 CONTINUE
C
   60 FORMAT(' ',1X)
      CALL ELAPSE(ICPT)
C
      RETURN
      END
C
C
C
      SUBROUTINE GTA(M,N,T,TIME,S,ICPT)
```

GTA

TIME, M

```
C
      INTEGER S(16),T(16,16),TIME,DIV,SUM(16),A(16)
      DIMENSION ITOT(16),X(16)
C
      CALL ELAPSE(ICPT)
C
C     CLEAR ARRAYS A AND X.
C
      DO 740 I=1,N
         A(I)=0
         X(I)=0
  740 CONTINUE
C
C     COMPUTE FUNCTION VALUE FOR EACH JOB AND STORE IN ARRAY X.
C
      DO 760 I=1,N
         IF (T(I,M) .LE. T(I,1)) THEN
            A(I)=1
         ELSE
            A(I)=-1
         ENDIF
         ITOT(1)=T(I,1)+T(I,2)
         DIV=ITOT(1)
C
         DO 750 K=2,M-1
            ITOT(K)=T(I,K)+T(I,K+1)
            IF (ITOT(K) .LT. DIV) DIV=ITOT(K)
  750    CONTINUE
C
         X(I)=FLOAT(A(I))/FLOAT(DIV)
  760 CONTINUE
C
C     SORT ARRAY X IN ASCENDING ORDER, BREAKING TIES WITH
C     SMALLEST TOTAL PROCESSING TIME.
C
      DO 775 I=1,N
         SUM(I)=0
         DO 770 J=1,M
  770       SUM(I)=SUM(I)+T(I,J)
  775 CONTINUE
C
      DO 790 K=1,N
         SMALL=10.
         DO 780 I=1,N
            IF (X(I) .LT. SMALL) THEN
               I3=I
               SMALL=X(I)
            ELSE IF (X(I) .EQ. SMALL .AND. SUM(I) .LT. SUM(I3)) THEN
               I3=I
               SMALL=X(I)
            ENDIF
  780    CONTINUE
         S(K)=I3
         X(I3)=12.
  790 CONTINUE
C
C     COMPUTE MAKESPAN OF RESULTING SEQUENCE.
C
      CALL MKSP(M,N,T,S,TIME)
C
```

```
      CALL ELAPSE(ICPT)
C
      RETURN
      END
C
C
C                                                                    DRA
      SUBROUTINE DRA(M,N,T,MIN,S,ICPT)
C
      INTEGER A(16),B(16),S(16),T(16,16),TIME
C
      CALL ELAPSE(ICPT)
C
C     CLEAR ARRAYS A AND B.
C
      DO 550 I=1,N
         A(I)=0
         B(I)=0
  550 CONTINUE
C
C     COMPUTE ARTIFICIAL TWO MACHINE TIMES FOR EACH JOB.
C
      DO 560 I=1,N
      DO 560 J=1,M
         A(I)=A(I)+(M-J+1)*(T(I,J))
         B(I)=B(I)+J*T(I,J)
  560 CONTINUE
C
C     APPLY JOHNSON'S RULE TO ARTIFICIAL PROBLEM
C
      CALL JOHN(A,B,S,N)
C
C     COMPUTE MAKESPAN ON RESULTING SEQUENCE.
C
      CALL MKSP(M,N,T,S,TIME)
      MIN=TIME
C
      CALL ELAPSE(ICPT)
C
      RETURN
      END
C
C
C                                                                    NEH
      SUBROUTINE NEH(M,N,T,MIN,S,TIME)
C
      INTEGER S(16),T(16,16),TIME,ST(16,16),TEMP(16),REL(16)
      INTEGER REL1(16),REL2(16),SUM(16),TIME1,TIME2,TAA,TAB
C
      CALL ELAPSE(ICPT)
C
C     CLEAR THE SUM ARRAY AND SUM THE PROCESSING TIMES FOR
C     EACH JOB.
C
      DO 600 I=1,N
  600    SUM(I)=0
C
      DO 610 I=1,N
         DO 605 J=1,M
  605    SUM(I)=SUM(I)+T(I,J)
```

```
  610 CONTINUE
C
C     SORT SUMS IN DESCENDING ORDER AND PUT ASSOCIATED JOB
C     NUMBER IN TEMP ARRAY.
C
      DO 620 K=1,N
         LARGE=0
         DO 615 I=1,N
            IF (SUM(I) .GE. LARGE) THEN
               I3=I
               LARGE=SUM(I)
            ENDIF
  615    CONTINUE
         TEMP(K)=I3
         SUM(I3)=0
  620 CONTINUE
C
C     DETERMINE BEST SEQUENCE OF FIRST TWO JOBS FROM
C     TEMP ARRAY.
C
      REL(1)=TEMP(1)
      REL(2)=TEMP(2)
C
      DO 645 I=1,2
         IF (I .EQ. 1) THEN
            REL1(1)=TEMP(1)
            REL1(2)=TEMP(2)
         ELSE
            REL1(1)=TEMP(2)
            REL1(2)=TEMP(1)
         ENDIF
C
C     CLEAR THE ST MATRIX.
C
         DO 625 IX=1,2
            DO 625 J=1,M
  625    ST(IX,J)=0
C
         ST(2,1)=ST(1,1)+T(REL1(1),1)
         DO 635 J=2,M
  635       ST(1,J)=ST(1,J-1)+T(REL1(1),J-1)
         DO 640 J=2,M
            TAA = ST(1,J) + T(REL1(1),J)
            TAB = ST(2,J-1) + T(REL1(2),J-1)
            ST(2,J) = MAXO(TAA,TAB)
  640    CONTINUE
C
         TIME=ST(2,M)+T(REL1(2),M)
            IF (I .EQ. 1) THEN
               TIME1=TIME
            ELSE
               TIME2=TIME
            ENDIF
C
  645 CONTINUE
C
      IF (TIME1 .LE. TIME2) THEN
         REL(1)=TEMP(1)
         REL(2)=TEMP(2)
      ELSE
```

```
         REL(1)=TEMP(2)
         REL(2)=TEMP(1)
      ENDIF
C
C     TAKE NEXT JOB FROM ORDERED SUMS AND INSERT IT INTO EACH
C     POSSIBLE POSITION IN THE PARTIAL SEQUENCE.  RETAIN
C     PARTIAL SEQUENCE WITH SHORTEST MAKESPAN.  REPEAT UNTIL
C     ALL JOBS ARE ASSIGNED TO A SEQUENCE POSITION.
C
      DO 685 I=3,N
C
      MIN=100000
C
            DO 650 I3=2,I
  650       REL1(I3)=REL(I3-1)
C
         REL1(1) = TEMP(I)
C
            CALL MKSP(M,I,T,REL1,TIME)
C
            IF (TIME .LT. MIN) THEN
            DO 660 II=1,I
               REL(II)=REL1(II)
  660       CONTINUE
            MIN=TIME
         ENDIF
      DO 675 K=1,I-1
         REL1(K)=REL1(K+1)
         REL1(K+1)=TEMP(I)
         CALL MKSP(M,I,T,REL1,TIME)
         IF (TIME .LT. MIN) THEN
            DO 680 II=1,I
            REL(II) = REL1(II)
  680       CONTINUE
            MIN = TIME
         ENDIF
  675 CONTINUE
C
  685 CONTINUE
C
C     PUT BEST SEQUENCE INTO ARRAY S FOR OUTPUT.
C
      DO 690 I=1,N
  690 S(I)=REL(I)
C
      CALL ELAPSE(ICPT)
C
      RETURN
      END
C
C
C                                                                    RDM
      SUBROUTINE RDM(M,N,T,MIN,S,ICPT)
C
      INTEGER T(16,16),S(16),A(16),B(16),Q,TIME
C
      CALL ELAPSE(ICPT)
C
C     SET NUMBER OF RANDOM SEQUENCES TO BE GENERATED BASED
C     ON PROBLEM SIZE.
```

```
C
      IF (N .EQ. 4) THEN
         NO = 10
         SEED=810312
      ELSE IF (N .EQ. 8) THEN
         NO = 400
         SEED=449503
      ELSE IF (N .EQ. 12) THEN
         NO = 1500
         SEED = 953413
      ELSE
         NO = 2000
         SEED = 101592
      ENDIF
C
C     GENERATE FIRST NUMBER IN SEQUENCE.
C
      RNDM = RANF(SEED)
C
C     ESTABLISH LARGE VALUE OF MINIMUM MAKESPAN.
C
      MIN=100000
C
C     ESTABLISH OUTER LOOP FOR NO ITERATIONS.
C
      DO 830 K=1,NO
C
C     PUT NUMBERS 1 THROUGH N IN ARRAY A.
C
         DO 800 I=1,N
  800       A(I)=I
C
C     GENERATE A RANDOM SEQUENCE INTO ARRAY B.
C
         Q = N
         DO 820 I=1,N-1
            RNDM=RANF(0)
            NUM=Q*RNDM+1
            B(I)=A(NUM)
            IF (Q .EQ. 2 .AND. NUM .EQ. 2) GOTO 820
               DO 810 II=NUM,Q-1
  810             A(II)=A(II+1)
         Q=Q-1
  820    CONTINUE
         B(N)=A(1)
C
C     CALCULATE MAKESPAN FOR CURRENT SEQUENCE.
C
      CALL MKSP(M,N,T,B,TIME)
C
C     COMPARE MAKESPAN FOR CURRENT SEQUENCE TO PREVIOUS
C     MINIMUM AND KEEP SMALLER OF THE TWO.
C
      IF (TIME .LT. MIN) THEN
         MIN = TIME
         DO 827 I=1,N
            S(I) = B(I)
  827    CONTINUE
      ENDIF
  830 CONTINUE
```

```
C
      CALL ELAPSE(ICPT)
C
      RETURN
      END
C
C                                                                    JOHN
C
      SUBROUTINE JOHN(A,B,S,N)
C
      INTEGER A(16), B(16), JOB(16), S(16)
C
C     CLEAR JOB ASSIGNMENT STATUS ARRAY.
C
      DO 400 I=1,N
  400    JOB(I)=0
C
C     RESET AVAILABLE SEQUENCE POSITIONS TO FIRST AND LAST.
C
      N1=1
      N2=N
C
C     OUTER LOOP TO PROCESS N JOBS
C
      DO 420 K=1,N
C
C     SET MINIMUM PROCESSING TIMES TO HIGH VALUE.
C
      MIN1=20000
      MIN2=20000
C
C     PASS OVER ANY JOB ALREADY ASSIGNED TO A SEQUENCE POSITION.
C
         DO 410 I=1,N
            IF (JOB(I) .GT. 0) GOTO 410
C
C     FIND SPT ON EACH MACHINE AND SET INDICES.
C
            IF (A(I) .LT. MIN1) THEN
               I1=I
               MIN1=A(I)
            ENDIF
C
            IF (B(I) .LT. MIN2) THEN
               I2=I
               MIN2=B(I)
            ENDIF
  410    CONTINUE
C
C     IF SPT IS ON MACHINE 1, ASSIGN JOB TO FIRST AVAILABLE
C     POSITION, OTHERWISE TO LAST AVAILABLE POSITION.
C     SET JOB ARRAY VALUE GREATER THAN ZERO.
C     CHANGE AVAILABLE POSITION INDICATOR.
C
         IF (A(I1) .LE. B(I2)) THEN
            S(N1)=I1
            JOB(I1)=10
            N1=N1+1
         ELSE
            S(N2)=I2
```

```
            JOB(I2)=10
            N2=N2-1
         ENDIF
C
  420 CONTINUE
C
      RETURN
      END
C
C
C
      SUBROUTINE MKSP(M,N,T,S,TIME)
C
      INTEGER TIME, T(16,16), S(16), ST(16,16),TAA,TAB
C
C     SET START TIME ARRAY ELEMENTS TO ZERO.
C
      DO 500 I=1,N
         DO 500 J=1,M
  500 ST(I,J)=0
C
C     COMPUTE STARTING TIMES OF EACH JOB ON MACHINE 1.
C
      DO 510 I=2,N
  510    ST(I,1)=ST(I-1,1) + T(S(I-1),1)
C
C     COMPUTE STARTING TIME OF FIRST JOB ON MACHINES 2 - M.
C
      DO 520 J=2,M
  520    ST(1,J)=ST(1,J-1) + T(S(1),J-1)
C
C     COMPUTE OTHER STARTING TIMES AS LARGER OF COMPLETION
C     OF SAME JOB ON PREVIOUS MACHINE OR COMPLETION OF
C     PREVIOUS JOB ON SAME MACHINE.
C
      DO 530 I=2,N
         DO 530 J=2,M
            TAA=ST(I,J-1) + T(S(I),J-1)
            TAB=ST(I-1,J) + T(S(I-1),J)
            IF (TAA .GE. TAB) THEN
               ST(I,J)=TAA
            ELSE
               ST(I,J)=TAB
            ENDIF
  530 CONTINUE
C
C     COMPUTE MAKESPAN AS START TIME OF LAST JOB ON LAST
C     MACHINE PLUS ITS PROCESSING TIME.
C
      TIME=ST(N,M) + T(S(N),M)
C
      RETURN
      END
```

```
C
C     FIRST IMPROVEMENT SEARCH ROUTINES
C
C     THIS PROGRAM READS THE PROBLEM SIZE AND PROCESSING TIME
C     MATRIX FROM THE PROBLEM GENERATION FILE.  IT THEN READS
C     THE DATA FROM THE HEURISTIC OUTPUT FILE.  IT APPLIES
C     EACH SEARCH ROUTINE TO THE HEURISTICALLY GENERATED
C     SEQUENCE AND TO THE ORDINAL SEQUENCE AND OUTPUTS THE
C     BEST SEQUENCE FOUND, ITS MAKESPAN AND THE COMBINED
C     COMPUTER PROCESSING TIME.  THIS VERSION USES A FIRST
C     IMPROVEMENT RULE IN EACH SEARCH ROUTINE.
C
      INTEGER T(16,16), SEQ(16), TIME, X
      CHARACTER MNE*3, SRMNE*5
C
C     READ PROBLEM SIZE FROM PROBLEM GENERATION FILE.
C
    5 READ(8,40,END=100) N,M,X
      WRITE(10,35) N,M
C
C     READ PROBLEM SIZE FROM HEURISTIC OUTPUT FILE
C
      READ(9,70) IN,IM
C
C     READ PROCESSING TIME MATRIX
C
      DO 30 K=1,10
         DO 10 I=1,N
   10    READ(8,45) (T(I,J),J=1,M)
      READ(8,50)
      DO 15 LL=1,6
C
C     READ HEURISTIC DATA FROM HEURISTIC OUTPUT FILE
C
      READ(9,60) MNE,L,TIME,ICPT,(SEQ(I),I=1,N)
      CALL SEARCH(MNE,K,TIME,ICPT,SEQ,T,M,N)
   15 CONTINUE
      READ(9,50)
C
C     GENERATE ORDINAL SEQUENCE TO INITIATE SEARCH ROUTINES
C
      DO 20 I=1,N
   20    SEQ(I)=I
      CALL MKSP(M,N,T,SEQ,TIME)
      MNE='ORD'
      ICPT = 0
      CALL SEARCH(MNE,K,TIME,ICPT,SEQ,T,M,N)
   30 CONTINUE
      GOTO 5
C
   35 FORMAT(1X,2I3)
   40 FORMAT(I4,I3,I7)
   45 FORMAT(I4,15I3)
   50 FORMAT(1X)
```

```
   60 FORMAT(1X,A4,I3,I5,I5,16I2)
   70 FORMAT(1X,2I3)
C
  100 STOP
      END
C
C
C
      SUBROUTINE SEARCH(MNE,K,TIME,ICPT,SEQ,T,M,N)
C
      INTEGER T(16,16), S(16), TIME, TCPT, SEQ(16)
      CHARACTER MNE*3, SRMNE*5
C
      ITIME = TIME
C
      SRMNE='FADJD'
      CALL FADJD(M,N,S,T,TIME,ICPT,TCPT,SEQ,ITIME)
      WRITE(10,200) MNE,SRMNE,K,TIME,TCPT,(S(I),I=1,N)
C
      SRMNE='FADJP'
      CALL FADJP(M,N,S,T,TIME,ICPT,TCPT,SEQ,ITIME)
      WRITE(10,200) MNE,SRMNE,K,TIME,TCPT,(S(I),I=1,N)
C
      SRMNE='FALLP'
      CALL FALLP(M,N,S,T,TIME,ICPT,TCPT,SEQ,ITIME)
      WRITE(10,200) MNE,SRMNE,K,TIME,TCPT,(S(I),I=1,N)
C
      SRMNE = 'FISGL'
      CALL FISGL(M,N,S,T,TIME,ICPT,TCPT,SEQ,ITIME)
      WRITE(10,200) MNE,SRMNE,K,TIME,TCPT,(S(I),I=1,N)
C
      SRMNE='FIAJP'
      CALL FIAJP(M,N,S,T,TIME,ICPT,TCPT,SEQ,ITIME)
      WRITE(10,200) MNE,SRMNE,K,TIME,TCPT,(S(I),I=1,N)
C
      SRMNE='FIALP'
      CALL FIALP(M,N,S,T,TIME,ICPT,TCPT,SEQ,ITIME)
      WRITE(10,200) MNE,SRMNE,K,TIME,TCPT,(S(I),I=1,N)
C
      WRITE(10,210)
C
  200 FORMAT(1X,A4,A6,I3,I5,I5,1X,16I2)
  210 FORMAT(1X)
C
      RETURN
      END
C
C
C
      SUBROUTINE FADJD(M,N,S,T,TIME,ICPT,TCPT,SEQ,ITIME)
C
      INTEGER S(16), T(16,16), TEMP(16), TIME, A(16), TCPT, SEQ(16)
C
      TCPT=ICPT
      CALL ELAPSE(JCPT)
C
C     SET MINIMUM TIME EQUAL TO HEURISTIC MAKESPAN
C
      MIN=ITIME
      DO 1212 I = 1,N
```

```
 1212 S(I) = SEQ(I)
C
C     PUT HEURISTIC SEQUENCE INTO HOLDING ARRAY
C
 1200 DO 1210 I=1,N
 1210    A(I)=S(I)
C
C     SET VALUE OF LAST MINIMUM
C
      LMIN = MIN
C
C     SWITCH ADJACENT DOUBLETS AND COMPUTE MAKESPAN.  IF NEW
C     SEQUENCE IMPROVES MAKESPAN, SET NEW MINIMUM TIME AND PUT
C     NEW SEQUENCE INTO SEQUENCE ARRAY AND BEGIN NEW SEARCH.
C
      DO 1230 I1=1,N-3
         DO 1215 I=1,N
 1215    TEMP(I)=A(I)
         ITEMP=TEMP(I1)
         TEMP(I1)=TEMP(I1+2)
         TEMP(I1+2)=ITEMP
         ITEMP=TEMP(I1+1)
         TEMP(I1+1)=TEMP(I1+3)
         TEMP(I1+3)=ITEMP
C
         CALL MKSP(M,N,T,TEMP,TIME)
C
         IF (TIME .LT. MIN) THEN
            MIN=TIME
            DO 1220 KK=1,N
 1220       S(KK)=TEMP(KK)
            GOTO 1200
         ENDIF
 1230 CONTINUE
C
C     CHECK CURRENT MINIMUM FOR IMPROVEMENT OVER LAST MINIMUM
C     AND RECYCLE TO ANOTHER SEARCH IF IMPROVEMENT ACHIEVED.
C
      IF (MIN .LT. LMIN) GOTO 1200
C
C     IF NO IMPROVEMENT, CALL ELAPSE, SUM CPT, AND RETURN
C
      CALL ELAPSE(JCPT)
      TCPT=TCPT + JCPT
      TIME = MIN
C
      RETURN
      END
C
C
C
      SUBROUTINE FADJP(M,N,S,T,TIME,ICPT,TCPT,SEQ,ITIME)
C
      INTEGER S(16), T(16,16), TEMP(16), TIME, A(16), TCPT, SEQ(16)
C
      TCPT = ICPT
      CALL ELAPSE(JCPT)
C
C     SET MINIMUM TIME EQUAL TO HEURISTIC MAKESPAN.
C
```

```
      MIN=ITIME
      DO 1002 I = 1,N
 1002 S(I) = SEQ(I)
C
C     PUT HEURISTIC SEQUENCE INTO HOLDING ARRAY.
C
 1000 DO 1005 I=1,N
 1005    A(I)=S(I)
C
C     SET INITIAL POINTERS AND LAST MINIMUM.
C
      I1=1
      I2=2
      LMIN=MIN
C
      DO 1020 I=1,N-1
C
C     RESTORE TEMPORARY ARRAY
C
         DO 1007 KK=1,N
 1007    TEMP(KK)=A(KK)
C
C     REVERSE ELEMENTS AT CURRENT POINTERS IN TEMPORARY ARRAY.
C
         ITEMP=TEMP(I1)
         TEMP(I1)=TEMP(I2)
         TEMP(I2)=ITEMP
C
         CALL MKSP(M,N,T,TEMP,TIME)
C
C     COMPARE MAKESPAN TO PREVIOUS MINIMUM AND KEEP BEST.
C     IF NEW SEQUENCE BETTER, PUT INTO TEMPORARY ARRAY.
C     RECYCLE TO NEW SEARCH.
C
         IF (TIME .LT. MIN) THEN
            MIN=TIME
            DO 1010 KK=1,N
 1010       S(KK)=TEMP(KK)
            GOTO 1000
         ENDIF
C
         I1=I1+1
         I2=I2+1
 1020 CONTINUE
C
      IF (MIN .LT. LMIN) GOTO 1000
C
      CALL ELAPSE(JCPT)
      TCPT = TCPT + JCPT
      TIME = MIN
C
      RETURN
      END
C
C
C
      SUBROUTINE FALLP(M,N,S,T,TIME,ICPT,TCPT,SEQ,ITIME)
C
      INTEGER S(16), T(16,16), TEMP(16), TIME, A(16), TCPT, SEQ(16)
C
```

```
      TCPT = ICPT
      CALL ELAPSE(JCPT)
C
C     SET MINIMUM TIME EQUAL TO HEURISTIC MAKESPAN.
C
      MIN = ITIME
      DO 1105 I = 1,N
 1105 S(I) = SEQ(I)
C
C     PUT HEURISTIC SEQUENCE INTO HOLDING ARRAY.
C
 1100 DO 1110 I=1,N
 1110    A(I) = S(I)
C
C     SET VALUE OF LAST MINIMUM
C
      LMIN = MIN
C
C     SWITCH ALL PAIRS TO RIGHT OF FIRST POINTER.  TEST FOR
C     MAKESPAN IMPROVEMENT AFTER EACH SWITCH.
C
      DO 1140 I1=1,N-1
         DO 1130 I2=I1+1,N
            DO 1115 I=1,N
 1115       TEMP(I) = A(I)
C
C     REVERSE ELEMENTS AT CURRENT POINTERS IN TEMP ARRAY.
C
            ITEMP = TEMP(I1)
            TEMP(I1) = TEMP(I2)
            TEMP(I2) = ITEMP
C
C     COMPUTE MAKESPAN AND COMPARE TO PREVIOUS MINIMUM.  IF
C     NEW SEQUENCE BETTER, PUT INTO SEQUENCE ARRAY AND RECYCLE
C     TO NEW SEARCH.
C
            CALL MKSP(M,N,T,TEMP,TIME)
            IF (TIME .LT. MIN) THEN
               MIN = TIME
               DO 1120 KK=1,N
 1120          S(KK) = TEMP(KK)
               GOTO 1100
            ENDIF
 1130    CONTINUE
 1140 CONTINUE
C
C     CHECK CURRENT MINIMUM FOR IMPROVEMENT OVER LAST MINIMUM
C     AND RECYCLE TO ANOTHER SWITCHING CYCLE IF IMPROVED.
C
      IF (MIN .LT. LMIN) GOTO 1100
C
      CALL ELAPSE(JCPT)
      TCPT = TCPT + JCPT
      TIME = MIN
C
      RETURN
      END
C
C
C
```

```
      SUBROUTINE FISGL(M,N,S,T,TIME,ICPT,TCPT,SEQ,ITIME)
C
      INTEGER S(16), T(16,16), TEMP(16), TIME, A(16), TCPT, SEQ(16)
C
      TCPT = ICPT
      CALL ELAPSE(JCPT)
C
C     SET MINIMUM TIME EQUAL TO HEURISTIC MAKESPAN.
C
      MIN = ITIME
      DO 1305 I = 1,N
 1305 S(I) = SEQ(I)
C
C     PUT HEURISTIC SEQUENCE INTO HOLDING ARRAY.
C
 1300 DO 1310 I=1,N
 1310 A(I) = S(I)
C
C     SET VALUE OF LAST MINIMUM.
C
      LMIN = MIN
C
      DO 1380 I=1,N
         DO 1320 K=1,N
 1320    TEMP(K) = A(K)
C
C     PUT ELEMENT TO BE INSERTED INTO TEMPORARY VARIABLE.
C
         ITEMP = TEMP(I)
         IF (I .GT. 1) THEN
C
C     MOVE ARRAY ELEMENTS TO LEFT OF I ONE SPACE TO RIGHT TO
C     OPEN UP FIRST ELEMENT. INSERT TEMP ELEMENT INTO FIRST
C     POSITION.
C
            DO 1330 II=I,2,-1
 1330       TEMP(II) = TEMP(II-1)
            TEMP(1) = ITEMP
C
C     COMPUTE MAKESPAN AND COMPARE TO PREVIOUS BEST.  IF NEW
C     SEQUENCE BETTER, PUT SEQUENCE INTO SEQUENCE ARRAY.
C
            CALL MKSP(M,N,T,TEMP,TIME)
            IF (TIME .LT. MIN) THEN
               MIN = TIME
               DO 1340 KK=1,N
 1340          S(KK) = TEMP(KK)
               GOTO 1300
            ENDIF
         ENDIF
C
C     MOVE NEXT ELEMENT ONE SPACE LEFT AND INSERT TEMPORARY
C     ELEMENT INTO VACATED SPACE.  COMPUTE MAKESPAN AND
C     COMPARE AS BEFORE.
C
         DO 1360 I3=2,N
            TEMP(I3-1) = TEMP(I3)
            TEMP(I3) = ITEMP
            IF (I3 .EQ. I) GOTO 1360
            CALL MKSP(M,N,T,TEMP,TIME)
```

```
            IF (TIME .LT. MIN) THEN
               MIN = TIME
               DO 1350 KK=1,N
 1350          S(KK) = TEMP(KK)
               GOTO 1300
            ENDIF
 1360    CONTINUE
 1380 CONTINUE
C
C     CHECK CURRENT MINIMUM FOR IMPROVEMENT OVER LAST MINIMUM
C     AND RECYCLE TO ANOTHER SEARCH IF IMPROVED.
C
      IF (MIN .LT. LMIN) GOTO 1300
C
      CALL ELAPSE(JCPT)
      TCPT = TCPT + JCPT
      TIME = MIN
C
      RETURN
      END
C
C
C
      SUBROUTINE FIAJP(M,N,S,T,TIME,ICPT,TCPT,SEQ,ITIME)
C
      INTEGER S(16), T(16,16), TEMP(16), TIME, A(16), TCPT, SEQ(16)
C
      TCPT = ICPT
      CALL ELAPSE(JCPT)
C
C     SET MINIMUM TIME EQUAL TO HEURISTIC MAKESPAN
C
      MIN = ITIME
      DO 1505 I = 1,N
 1505 S(I) = SEQ(I)
C
C     PUT HEURISTIC SEQUENCE INTO HOLDING ARRAY
C
 1500 DO 1510 I=1,N
 1510 A(I) = S(I)
C
C     SET VALUE OF LAST MINIMUM
C
      LMIN = MIN
C
      DO 1580 I=1,N-1
C
C     SET TEMPORARY ARRAY EQUAL TO HOLDING ARRAY
C
         DO 1520 K=1,N
 1520    TEMP(K) = A(K)
C
C     PUT ELEMENTS TO BE INSERTED INTO TEMPORARY VARIABLES
C
         ITEMP = TEMP(I)
         JTEMP = TEMP(I+1)
C
C     IF I > 1, MOVE ARRAY ELEMENTS TO THE LEFT OF I TWO SPACES
C     TO RIGHT TO OPEN UP ELEMENTS 1 AND 2.
C
```

```
         IF (I .GT. 1) THEN
            DO 1530 II = I,2,-1
 1530       TEMP(II+1) = TEMP(II-1)
C
C     INSERT TEMPORARY ELEMENTS IN FIRST TWO POSITIONS
C
            TEMP(1) = ITEMP
            TEMP(2) = JTEMP
C
C     COMPUTE MAKESPAN AND COMPARE WITH PREVIOUS BEST.  IF NEW
C     SEQUENCE BETTER, UPDATE MINIMUM AND SEQUENCE ARRAY.
C
            CALL MKSP(M,N,T,TEMP,TIME)
            IF (TIME .LT. MIN) THEN
               MIN = TIME
               DO 1540 K=1,N
 1540          S(K) = TEMP(K)
               GOTO 1500
            ENDIF
         ENDIF
C
C     MOVE NEXT ELEMENT TWO SPACES LEFT AND INSERT TEMPORARY
C     ELEMENTS INTO VACATED SPACES.  COMPUTE MAKESPAN AND
C     COMPARE AS BEFORE.
C
         DO 1560 I3=2,N-1
            TEMP(I3-1) = TEMP(I3+1)
            TEMP(I3) = ITEMP
            TEMP(I3+1) = JTEMP
            IF (I3 .EQ. I) GOTO 1560
            CALL MKSP(M,N,T,TEMP,TIME)
            IF (TIME .LT. MIN) THEN
               MIN = TIME
               DO 1550 K=1,N
 1550          S(K) = TEMP(K)
               GOTO 1500
            ENDIF
 1560    CONTINUE
 1580 CONTINUE
C
      IF (MIN .LT. LMIN) GOTO 1500
C
      CALL ELAPSE(JCPT)
      TCPT = TCPT + JCPT
      TIME = MIN
C
      RETURN
      END
C
C
C
      SUBROUTINE FIALP(M,N,S,T,TIME,ICPT,TCPT,SEQ,ITIME)
C
      INTEGER S(16), T(16,16), TEMP(16), TIME, A(16), TCPT, SEQ(16)
C
      TCPT = ICPT
      CALL ELAPSE(JCPT)
C
C     SET MINIMUM TIME EQUAL TO HEURISTIC MAKESPAN, PUT
C     HEURISTIC SEQUENCE INTO HOLDING ARRAY, AND SET VALUE
```

```
C      OF LAST MINIMUM.
C
       MIN = ITIME
       DO 1705 I = 1,N
 1705 S(I) = SEQ(I)
C
 1700 DO 1710 I=1,N
 1710 A(I) = S(I)
       LMIN = MIN
C
C      SET UP MAIN LOOP.
C
       DO 1780 I=1,N-1
          DO 1770 J=I+1,N
C
C      SET TEMP ARRAY EQUAL TO HOLDING ARRAY.
C
             DO 1720 K=1,N
 1720        TEMP(K) = A(K)
C
C      WITH INITIAL SEQUENCE IN PLACE, ON FIRST TIME THROUGH
C      J-LOOP, REVERSE POSITIONS OF TEMP(I) AND TEMP(J).
C
             IF (J .EQ. I+1) THEN
                TEMP(I) = A(J)
                TEMP(J) = A(I)
C
C      COMPUTE MAKESPAN, COMPARE, AND UPDATE SEQUENCE ARRAY.
C
                CALL MKSP(M,N,T,TEMP,TIME)
C
                IF (TIME .LT. MIN) THEN
                   MIN = TIME
                   DO 1730 K=1,N
 1730              S(K) = TEMP(K)
                   GOTO 1700
                ENDIF
             ENDIF
C
C      IF I>1 OR J>I+1, RIGHT JUSTIFY TEMPORARY ARRAY TO OPEN
C      FIRST TWO POSITIONS AND INSERT TEMPORARY ELEMENTS.
C
          IF (I .GT. 1 .OR. J .GT. I+1) THEN
             TEMP(1) = A(I)
             A(I) = 0
             TEMP(2) = A(J)
             A(J) = 0
             ITOP = N
             DO 1740 L= N,1,-1
                IF (A(L) .GT. 0) THEN
                   TEMP(ITOP) = A(L)
                   ITOP = ITOP-1
                ENDIF
 1740        CONTINUE
C
C      RESTORE HOLDING ARRAY.
C
             A(I) = TEMP(1)
             A(J) = TEMP(2)
C
```

```
C      COMPUTE MAKESPAN OF PRIMARY SEQUENCE AND COMPARE WITH
C      PREVIOUS BEST.  KEEP BEST AND UPDATE MINIMUM AND SEQUENCE
C      ARRAY AS NECESSARY.
C
             CALL MKSP(M,N,T,TEMP,TIME)
C
             IF (TIME .LT. MIN) THEN
                MIN = TIME
                DO 1745 K=1,N
 1745           S(K) = TEMP(K)
                GOTO 1700
             ENDIF
C
C      REVERSE SEQUENCE OF ITEMP AND JTEMP.  COMPUTE MAKESPAN AND
C      COMPARE AS BEFORE.
C
             TEMP(1) = A(J)
             TEMP(2) = A(I)
C
             CALL MKSP(M,N,T,TEMP,TIME)
C
             IF (TIME .LT. MIN) THEN
                MIN = TIME
                DO 1750 K=1,N
 1750           S(K) = TEMP(K)
                GOTO 1700
             ENDIF
C
C      SHUFFLE INSERT ELEMENTS ONE SPACE TO RIGHT.  TEST PRIMARY
C      AND REVERSED SEQUENCES.
C
             DO 1765 II=3,N
                TEMP(II-2) = TEMP(II)
                TEMP(II-1) = A(I)
                TEMP(II) = A(J)
                IF (TEMP(I) .EQ. A(I) .AND. TEMP(J) .EQ. A(J))
     1             GOTO 1765
C
C      TEST AND COMPARE PRIMARY SEQUENCE
C
             CALL MKSP(M,N,T,TEMP,TIME)
             IF (TIME .LT. MIN) THEN
                MIN = TIME
                DO 1755 K=1,N
 1755           S(K) = TEMP(K)
                GOTO 1700
             ENDIF
C
C      TEST AND COMPARE REVERSED SEQUENCE.
C
             TEMP(II-1) = A(J)
             TEMP(II) = A(I)
C
             CALL MKSP(M,N,T,TEMP,TIME)
             IF (TIME .LT. MIN) THEN
                MIN = TIME
                DO 1760 K=1,N
 1760           S(K) = TEMP(K)
                GOTO 1700
             ENDIF
```

```
 1765         CONTINUE
              ENDIF
 1770      CONTINUE
 1780 CONTINUE
C
      IF (MIN .LT. LMIN) GOTO 1700
C
      CALL ELAPSE(JCPT)
      TCPT = TCPT + JCPT
      TIME = MIN
C
      RETURN
      END
C
C
C
      SUBROUTINE MKSP(M,N,T,S,TIME)
C
      INTEGER TIME, T(16,16), S(16), ST(16,16)
C
C     SET START TIME ARRAY ELEMENTS TO 0.
C
      DO 500 I=1,N
         DO 500 J=1,M
  500 ST(I,J) = 0
C
C     COMPUTE STARTING TIME OF EACH JOB ON MACHINE 1.
C
      DO 510 I=2,N
  510 ST(I,1) = ST(I-1,1) + T(S(I-1),1)
C
C     COMPUTE START TIME OF JOB 1 ON MACHINES 2 - M.
C
      DO 520 J=2,M
  520 ST(1,J) = ST(1,J-1) + T(S(1),J-1)
C
C     COMPUTE OTHER START TIMES AS LARGER OF COMPLETION OF SAME
C     JOB ON PREVIOUS MACHINE OR COMPLETION OF PREVIOUS JOB ON
C     SAME MACHINE.
C
      DO 530 I=2,N
         DO 530 J=2,M
            TAA = ST(I,J-1) + T(S(I),J-1)
            TAB = ST(I-1,J) + T(S(I-1),J)
            IF (TAA .GE. TAB) THEN
               ST(I,J) = TAA
            ELSE
               ST(I,J) = TAB
            ENDIF
  530 CONTINUE
C
C     COMPUTE MAKESPAN AS START TIME OF LAST JOB ON LAST MACHINE
C     PLUS ITS PROCESSING TIME.
C
      TIME = ST(N,M) + T(S(N),M)
C
      RETURN
      END
```

```
C
C     BEST IMPROVEMENT SEARCH ROUTINES
C
C     THIS PROGRAM READS THE PROBLEM SIZE AND PROCESSING TIME
C     MATRIX FROM THE PROBLEM GENERATION FILE.  IT THEN READS
C     THE DATA FROM THE HEURISTIC OUTPUT FILE.  IT APPLIES
C     EACH SEARCH ROUTINE TO THE HEURISTICALLY GENERATED
C     SEQUENCE AND TO THE ORDINAL SEQUENCE AND OUTPUTS THE
C     BEST SEQUENCE FOUND, ITS MAKESPAN AND THE COMBINED
C     COMPUTER PROCESSING TIME.  THIS VERSION USES A BEST
C     IMPROVEMENT RULE IN EACH SEARCH ROUTINE.
C
      INTEGER T(16,16), SEQ(16), TIME, X
      CHARACTER MNE*3, SRMNE*5
C
C     READ PROBLEM SIZE FROM PROBLEM GENERATION FILE.
C
    5 READ(8,40,END=100) N,M,X
      WRITE(10,35) N,M
C
C     READ PROBLEM SIZE FROM HEURISTIC INPUT FILE
C
      READ(9,70) IN,IM
C
C     READ PROCESSING TIME MATRIX
C
      DO 30 K=1,10
         DO 10 I=1,N
   10    READ(8,45) (T(I,J),J=1,M)
      READ(8,50)
      DO 15 LL=1,6
C
C     READ HEURISTIC DATA FROM HEURISTIC OUTPUT FILE
C
      READ(9,60) MNE,L,TIME,ICPT,(SEQ(I),I=1,N)
      CALL SEARCH(MNE,K,TIME,ICPT,SEQ,T,M,N)
   15 CONTINUE
      READ(9,50)
C
C     GENERATE ORDINAL SEQUENCE TO INITIATE SEARCH ROUTINES
C
      DO 20 I=1,N
   20    SEQ(I)=I
      CALL MKSP(M,N,T,SEQ,TIME)
      MNE='ORD'
      ICPT = 0
      CALL SEARCH(MNE,K,TIME,ICPT,SEQ,T,M,N)
   30 CONTINUE
      GOTO 5
C
   35 FORMAT(1X,2I3)
   40 FORMAT(I4,I3,I7)
   45 FORMAT(I4,15I3)
   50 FORMAT(1X)
```

```
   60 FORMAT(1X,A4,I3,I5,I5,16I2)
   70 FORMAT(1X,2I3)
C
  100 STOP
      END
C
C
C
      SUBROUTINE SEARCH(MNE,K,TIME,ICPT,SEQ,T,M,N)
C
      INTEGER T(16,16), S(16), TIME, TCPT, SEQ(16)
      CHARACTER MNE*3, SRMNE*5
C
      ITIME = TIME
C
      SRMNE='BADJD'
      CALL BADJD(M,N,S,T,TIME,ICPT,TCPT,SEQ,ITIME)
      WRITE(10,200) MNE,SRMNE,K,TIME,TCPT,(S(I),I=1,N)
C
      SRMNE='BADJP'
      CALL BADJP(M,N,S,T,TIME,ICPT,TCPT,SEQ,ITIME)
      WRITE(10,200) MNE,SRMNE,K,TIME,TCPT,(S(I),I=1,N)
C
      SRMNE='BALLP'
      CALL BALLP(M,N,S,T,TIME,ICPT,TCPT,SEQ,ITIME)
      WRITE(10,200) MNE,SRMNE,K,TIME,TCPT,(S(I),I=1,N)
C
      SRMNE='BISGL'
      CALL BISGL(M,N,S,T,TIME,ICPT,TCPT,SEQ,ITIME)
      WRITE(10,200) MNE,SRMNE,K,TIME,TCPT,(S(I),I=1,N)
C
      SRMNE='BIAJP'
      CALL BIAJP(M,N,S,T,TIME,ICPT,TCPT,SEQ,ITIME)
      WRITE(10,200) MNE,SRMNE,K,TIME,TCPT,(S(I),I=1,N)
C
      SRMNE='BIALP'
      CALL BIALP(M,N,S,T,TIME,ICPT,TCPT,SEQ,ITIME)
      WRITE(10,200) MNE,SRMNE,K,TIME,TCPT,(S(I),I=1,N)
C
      WRITE(10,210)
C
  200 FORMAT(1X,A4,A6,I3,I5,I5,1X,16I2)
  210 FORMAT(1X)
C
      RETURN
      END
C
C
C
      SUBROUTINE BADJD(M,N,S,T,TIME,ICPT,TCPT,SEQ,ITIME)
C
      INTEGER S(16), T(16,16), TEMP(16), TIME, A(16), TCPT, SEQ(16)
C
      TCPT=ICPT
      CALL ELAPSE(JCPT)
C
C     SET MINIMUM TIME EQUAL TO HEURISTIC MAKESPAN
C
      MIN=ITIME
      DO 1212 I = 1,N
```

```
 1212 S(I) = SEQ(I)
C
C     PUT HEURISTIC SEQUENCE INTO HOLDING ARRAY
C
 1200 DO 1210 I=1,N
 1210    A(I)=S(I)
C
C     SET VALUE OF LAST MINIMUM
C
      LMIN=MIN
C
C     SWITCH ADJACENT DOUBLETS AND COMPUTE MAKESPAN.  IF NEW
C     SEQUENCE IMPROVES MAKESPAN, SET NEW MINIMUM TIME AND PUT
C     NEW SEQUENCE INTO SEQUENCE ARRAY.
C
      DO 1230 I1=1,N-3
         DO 1215 I=1,N
 1215    TEMP(I)=A(I)
         ITEMP=TEMP(I1)
         TEMP(I1)=TEMP(I1+2)
         TEMP(I1+2)=ITEMP
         ITEMP=TEMP(I1+1)
         TEMP(I1+1)=TEMP(I1+3)
         TEMP(I1+3)=ITEMP
C
         CALL MKSP(M,N,T,TEMP,TIME)
C
         IF (TIME .LT. MIN) THEN
            MIN=TIME
            DO 1220 KK=1,N
 1220       S(KK)=TEMP(KK)
         ENDIF
 1230 CONTINUE
C
C     CHECK CURRENT MINIMUM FOR IMPROVEMENT OVER LAST MINIMUM
C     AND RECYCLE TO ANOTHER SEARCH IF IMPROVEMENT ACHIEVED.
C
      IF (MIN .LT. LMIN) GOTO 1200
C
C     IF NO IMPROVEMENT, CALL ELAPSE, SUM CPT, AND RETURN
C
      CALL ELAPSE(JCPT)
      TCPT=TCPT + JCPT
      TIME = MIN
C
      RETURN
      END
C
C
C
      SUBROUTINE BADJP(M,N,S,T,TIME,ICPT,TCPT,SEQ,ITIME)
C
      INTEGER S(16), T(16,16), TEMP(16), TIME, A(16), TCPT, SEQ(16)
C
      TCPT = ICPT
      CALL ELAPSE(JCPT)
C
C     SET MINIMUM TIME EQUAL TO HEURISTIC MAKESPAN.
C
      MIN=ITIME
```

```
      DO 1002 I = 1,N
 1002 S(I) = SEQ(I)
C
C     PUT HEURISTIC SEQUENCE INTO HOLDING ARRAY.
C
 1000 DO 1005 I=1,N
 1005    A(I)=S(I)
C
C     SET INITIAL POINTERS AND LAST MINIMUM.
C
      I1=1
      I2=2
      LMIN=MIN
C
      DO 1020 I=1,N-1
C
C     RESTORE TEMPORARY ARRAY
C
         DO 1007 KK=1,N
 1007    TEMP(KK)=A(KK)
C
C     REVERSE ELEMENTS AT CURRENT POINTERS IN TEMPORARY ARRAY.
C
         ITEMP=TEMP(I1)
         TEMP(I1)=TEMP(I2)
         TEMP(I2)=ITEMP
C
         CALL MKSP(M,N,T,TEMP,TIME)
C
C     COMPARE MAKESPAN TO PREVIOUS MINIMUM AND KEEP BEST.
C     IF NEW SEQUENCE BETTER, PUT INTO TEMPORARY ARRAY.
C
         IF (TIME .LT. MIN) THEN
            MIN=TIME
            DO 1010 KK=1,N
 1010       S(KK)=TEMP(KK)
         ENDIF
C
         I1=I1+1
         I2=I2+1
 1020 CONTINUE
C
      IF (MIN .LT. LMIN) GOTO 1000
C
      CALL ELAPSE(JCPT)
      TCPT = TCPT + JCPT
      TIME = MIN
C
      RETURN
      END
C
C
C
      SUBROUTINE BALLP(M,N,S,T,TIME,ICPT,TCPT,SEQ,ITIME)
C
      INTEGER S(16), T(16,16), TEMP(16), TIME, A(16), TCPT, SEQ(16)
C
      TCPT = ICPT
      CALL ELAPSE(JCPT)
C
```

```
C      SET MINIMUM TIME EQUAL TO HEURISTIC MAKESPAN.
C
       MIN = ITIME
       DO 1105 I = 1,N
 1105 S(I) = SEQ(I)
C
C      PUT HEURISTIC SEQUENCE INTO HOLDING ARRAY.
C
 1100 DO 1110 I=1,N
 1110    A(I) = S(I)
C
C      SET VALUE OF LAST MINIMUM
C
       LMIN = MIN
C
C      SWITCH ALL PAIRS TO RIGHT OF FIRST POINTER.  TEST FOR
C      MAKESPAN IMPROVEMENT AFTER EACH SWITCH.
C
       DO 1140 I1=1,N-1
          DO 1130 I2=I1+1,N
             DO 1115 I=1,N
 1115        TEMP(I) = A(I)
C
C      REVERSE ELEMENTS AT CURRENT POINTERS IN TEMP ARRAY.
C
             ITEMP = TEMP(I1)
             TEMP(I1) = TEMP(I2)
             TEMP(I2) = ITEMP
C
C      COMPUTE MAKESPAN AND COMPARE TO PREVIOUS MINIMUM.  IF
C      NEW SEQUENCE BETTER, PUT INTO SEQUENCE ARRAY.
C
             CALL MKSP(M,N,T,TEMP,TIME)
             IF (TIME .LT. MIN) THEN
                MIN = TIME
                DO 1120 KK=1,N
 1120           S(KK) = TEMP(KK)
             ENDIF
 1130     CONTINUE
 1140 CONTINUE
C
C      CHECK CURRENT MINIMUM FOR IMPROVEMENT OVER LAST MINIMUM
C      AND RECYCLE TO ANOTHER SWITCHING CYCLE IF IMPROVED.
C
       IF (MIN .LT. LMIN) GOTO 1100
C
       CALL ELAPSE(JCPT)
       TCPT = TCPT + JCPT
       TIME = MIN
C
       RETURN
       END
C
C
C
       SUBROUTINE BISGL(M,N,S,T,TIME,ICPT,TCPT,SEQ,ITIME)
C
       INTEGER S(16), T(16,16), TEMP(16), TIME, A(16), TCPT,SEQ(16)
C
       TCPT = ICPT
```

```
      CALL ELAPSE(JCPT)
C
C     SET MINIMUM TIME EQUAL TO HEURISTIC MAKESPAN.
C
      MIN = ITIME
      DO 1305 I = 1,N
 1305 S(I) = SEQ(I)
C
C     PUT HEURISTIC SEQUENCE INTO HOLDING ARRAY.
C
 1300 DO 1310 I=1,N
 1310 A(I) = S(I)
C
C     SET VALUE OF LAST MINIMUM.
C
      LMIN = MIN
C
      DO 1380 I=1,N
         DO 1320 K=1,N
 1320    TEMP(K) = A(K)
C
C     PUT ELEMENT TO BE INSERTED INTO TEMPORARY VARIABLE.
C
         ITEMP = TEMP(I)
         IF (I .GT. 1) THEN
C
C     MOVE ARRAY ELEMENTS TO LEFT OF I ONE SPACE TO RIGHT TO
C     OPEN UP FIRST ELEMENT. INSERT TEMP ELEMENT INTO FIRST
C     POSITION.
C
            DO 1330 II=I,2,-1
 1330       TEMP(II) = TEMP(II-1)
            TEMP(1) = ITEMP
C
C     COMPUTE MAKESPAN AND COMPARE TO PREVIOUS BEST.  IF NEW
C     SEQUENCE BETTER, PUT SEQUENCE INTO SEQUENCE ARRAY.
C
            CALL MKSP(M,N,T,TEMP,TIME)
            IF (TIME .LT. MIN) THEN
               MIN = TIME
               DO 1340 KK=1,N
 1340          S(KK) = TEMP(KK)
            ENDIF
         ENDIF
C
C     MOVE NEXT ELEMENT ONE SPACE LEFT AND INSERT TEMPORARY
C     ELEMENT INTO VACATED SPACE.  COMPUTE MAKESPAN AND
C     COMPARE AS BEFORE.
C
         DO 1360 I3=2,N
            TEMP(I3-1) = TEMP(I3)
            TEMP(I3) = ITEMP
            IF (I3 .EQ. I) GOTO 1360
            CALL MKSP(M,N,T,TEMP,TIME)
            IF (TIME .LT. MIN) THEN
               MIN = TIME
               DO 1350 KK=1,N
 1350          S(KK) = TEMP(KK)
            ENDIF
 1360    CONTINUE
```

```
 1380 CONTINUE
C
C      CHECK CURRENT MINIMUM FOR IMPROVEMENT OVER LAST MINIMUM
C      AND RECYCLE TO ANOTHER SEARCH IF IMPROVED.
C
       IF (MIN .LT. LMIN) GOTO 1300
C
       CALL ELAPSE(JCPT)
       TCPT = TCPT + JCPT
       TIME = MIN
C
       RETURN
       END
C
C
C
       SUBROUTINE BIAJP(M,N,S,T,TIME,ICPT,TCPT,SEQ,ITIME)
C
       INTEGER S(16), T(16,16), TEMP(16), TIME, A(16), TCPT, SEQ(16)
C
       TCPT = ICPT
       CALL ELAPSE(JCPT)
C
C      SET MINIMUM TIME EQUAL TO HEURISTIC MAKESPAN
C
       MIN = ITIME
       DO 1505 I = 1,N
 1505 S(I) = SEQ(I)
C
C      PUT HEURISTIC SEQUENCE INTO HOLDING ARRAY
C
 1500 DO 1510 I=1,N
 1510 A(I) = S(I)
C
C      SET VALUE OF LAST MINIMUM
C
       LMIN = MIN
C
       DO 1580 I=1,N-1
C
C      SET TEMPORARY ARRAY EQUAL TO HOLDING ARRAY
C
          DO 1520 K=1,N
 1520     TEMP(K) = A(K)
C
C      PUT ELEMENTS TO BE INSERTED INTO TEMPORARY VARIABLES
C
          ITEMP = TEMP(I)
          JTEMP = TEMP(I+1)
C
C      IF I > 1, MOVE ARRAY ELEMENTS TO THE LEFT OF I TWO SPACES
C      TO RIGHT TO OPEN UP ELEMENTS 1 AND 2.
C
          IF (I .GT. 1) THEN
             DO 1530 II = I,2,-1
 1530        TEMP(II+1) = TEMP(II-1)
C
C      INSERT TEMPORARY ELEMENTS IN FIRST TWO POSITIONS
C
             TEMP(1) = ITEMP
```

```
            TEMP(2) = JTEMP
C
C     COMPUTE MAKESPAN AND COMPARE WITH PREVIOUS BEST.  IF NEW
C     SEQUENCE BETTER, UPDATE MINIMUM AND SEQUENCE ARRAY.
C
            CALL MKSP(M,N,T,TEMP,TIME)
            IF (TIME .LT. MIN) THEN
               MIN = TIME
               DO 1540 K=1,N
 1540          S(K) = TEMP(K)
            ENDIF
         ENDIF
C
C     MOVE NEXT ELEMENT TWO SPACES LEFT AND INSERT TEMPORARY
C     ELEMENTS INTO VACATED SPACES.  COMPUTE MAKESPAN AND
C     COMPARE AS BEFORE.
C
         DO 1560 I3=2,N-1
            TEMP(I3-1) = TEMP(I3+1)
            TEMP(I3) = ITEMP
            TEMP(I3+1) = JTEMP
            IF (I3 .EQ. I) GOTO 1560
            CALL MKSP(M,N,T,TEMP,TIME)
            IF (TIME .LT. MIN) THEN
               MIN = TIME
               DO 1550 K=1,N
 1550          S(K) = TEMP(K)
            ENDIF
 1560    CONTINUE
 1580 CONTINUE
C
      IF (MIN .LT. LMIN) GOTO 1500
C
      CALL ELAPSE(JCPT)
      TCPT = TCPT + JCPT
      TIME = MIN
C
      RETURN
      END
C
C
C
      SUBROUTINE BIALP(M,N,S,T,TIME,ICPT,TCPT,SEQ,ITIME)
C
      INTEGER S(16), T(16,16), TEMP(16), TIME, A(16), TCPT, SEQ(16)
C
      TCPT = ICPT
      CALL ELAPSE(JCPT)
C
C     SET MINIMUM TIME EQUAL TO HEURISTIC MAKESPAN, PUT
C     HEURISTIC SEQUENCE INTO HOLDING ARRAY, AND SET VALUE
C     OF LAST MINIMUM.
C
      MIN = ITIME
      DO 1705 I = 1,N
 1705 S(I) = SEQ(I)
C
 1700 DO 1710 I=1,N
 1710 A(I) = S(I)
      LMIN = MIN
```

```
C
C     SET UP MAIN LOOP.
C
      DO 1780 I=1,N-1
         DO 1770 J=I+1,N
C
C     SET TEMP ARRAY EQUAL TO HOLDING ARRAY.
C
            DO 1720 K=1,N
 1720       TEMP(K) = A(K)
C
C     WITH INITIAL SEQUENCE IN PLACE, ON FIRST TIME THROUGH
C     J-LOOP, REVERSE POSITIONS OF TEMP(I) AND TEMP(J).
C
            IF (J .EQ. I+1) THEN
               TEMP(I) = A(J)
               TEMP(J) = A(I)
C
C     COMPUTE MAKESPAN, COMPARE, AND UPDATE SEQUENCE ARRAY.
C
               CALL MKSP(M,N,T,TEMP,TIME)
C
               IF (TIME .LT. MIN) THEN
                  MIN = TIME
                  DO 1730 K=1,N
 1730             S(K) = TEMP(K)
               ENDIF
            ENDIF
C
C     IF I>1 OR J>I+1, RIGHT JUSTIFY TEMPORARY ARRAY TO OPEN
C     FIRST TWO POSITIONS AND INSERT TEMPORARY ELEMENTS.
C
         IF (I .GT. 1 .OR. J .GT. I+1) THEN
            TEMP(1) = A(I)
            A(I) = 0
            TEMP(2) = A(J)
            A(J) = 0
            ITOP = N
            DO 1740 L= N,1,-1
               IF (A(L) .GT. 0) THEN
                  TEMP(ITOP) = A(L)
                  ITOP = ITOP-1
               ENDIF
 1740       CONTINUE
C
C     RESTORE HOLDING ARRAY.
C
            A(I) = TEMP(1)
            A(J) = TEMP(2)
C
C     COMPUTE MAKESPAN OF PRIMARY SEQUENCE AND COMPARE WITH
C     PREVIOUS BEST.  KEEP BEST AND UPDATE MINIMUM AND SEQUENCE
C     ARRAY AS NECESSARY.
C
            CALL MKSP(M,N,T,TEMP,TIME)
C
            IF (TIME .LT. MIN) THEN
               MIN = TIME
               DO 1745 K=1,N
 1745          S(K) = TEMP(K)
```

```
            ENDIF
C
C     REVERSE SEQUENCE OF TEMP ELEMENTS, COMPUTE MAKESPAN AND
C     COMPARE AS BEFORE.
C
            TEMP(1) = A(J)
            TEMP(2) = A(I)
C
            CALL MKSP(M,N,T,TEMP,TIME)
C
            IF (TIME .LT. MIN) THEN
               MIN = TIME
               DO 1750 K=1,N
 1750          S(K) = TEMP(K)
            ENDIF
C
C     SHUFFLE INSERT ELEMENTS ONE SPACE TO RIGHT.  TEST PRIMARY
C     AND REVERSED SEQUENCES.
C
            DO 1765 II=3,N
               TEMP(II-2) = TEMP(II)
               TEMP(II-1) = A(I)
               TEMP(II) = A(J)
               IF (TEMP(I) .EQ. A(I) .AND. TEMP(J) .EQ. A(J))
     1            GOTO 1765
C
C     TEST AND COMPARE PRIMARY SEQUENCE
C
            CALL MKSP(M,N,T,TEMP,TIME)
            IF (TIME .LT. MIN) THEN
               MIN = TIME
               DO 1755 K=1,N
 1755          S(K) = TEMP(K)
            ENDIF
C
C     TEST AND COMPARE REVERSED SEQUENCE.
C
            TEMP(II-1) = A(J)
            TEMP(II) = A(I)
C
            CALL MKSP(M,N,T,TEMP,TIME)
            IF (TIME .LT. MIN) THEN
               MIN = TIME
               DO 1760 K=1,N
 1760          S(K) = TEMP(K)
            ENDIF
 1765       CONTINUE
         ENDIF
 1770    CONTINUE
 1780 CONTINUE
C
      IF (MIN .LT. LMIN) GOTO 1700
C
      CALL ELAPSE(JCPT)
      TCPT = TCPT + JCPT
      TIME = MIN
C
      RETURN
      END
C
```

```
C
C
      SUBROUTINE MKSP(M,N,T,S,TIME)
C
      INTEGER TIME, T(16,16), S(16), ST(16,16)
C
C     SET START TIME ARRAY ELEMENTS TO ZERO.
C
      DO 500 I=1,N
         DO 500 J=1,M
  500 ST(I,J) = 0
C
C     COMPUTE STARTING TIME FOR EACH JOB ON MACHINE 1.
C
      DO 510 I=2,N
  510 ST(I,1) = ST(I-1,1) + T(S(I-1),1)
C
C     COMPUTE STARTING TIME FOR FIRST JOB ON MACHINES 2 - M.
C
      DO 520 J=2,M
  520 ST(1,J) = ST(1,J-1) + T(S(1),J-1)
C
C     COMPUTE OTHER STARTING TIMES AS LARGER OF COMPLETION
C     OF SAME JOB ON PREVIOUS MACHINE OR COMPLETION OF PREVIOUS
C     JOB ON SAME MACHINE.
C
      DO 530 I=2,N
         DO 530 J=2,M
            TAA = ST(I,J-1) + T(S(I),J-1)
            TAB = ST(I-1,J) + T(S(I-1),J)
            IF (TAA .GE. TAB) THEN
               ST(I,J) = TAA
            ELSE
               ST(I,J) = TAB
            ENDIF
  530 CONTINUE
C
C     COMPUTE MAKESPAN AS START TIME OF LAST JOB ON LAST
C     MACHINE PLUS ITS PROCESSING TIME.
C
      TIME = ST(N,M) + T(S(N),M)
C
      RETURN
      END
```

VITA

Francis Dean Booth

Candidate for the Degree of

Doctor of Philosophy

Thesis: AN EVALUATION OF FLOW SHOP SCHEDULING HEURISTICS

Major Field: Business Administration

Biographical:

Personal Data: Born in Marcus, Iowa, July 5, 1936, the son of Frank D. and Anna A. Booth. Married to Ruth E. Smith on June 8, 1958.

Education: Graduated from Redfield High School, Redfield, Iowa, in May, 1954; received Bachelor of Science in Business Administration degree from Simpson College in June 1958; received Master of Business Administration degree from Arkansas State University in May, 1983; completed requirements for the Doctor of Philosophy degree at Oklahoma State University in December, 1987.

Professional Experience: Graduate Assistant, Department of Management and Marketing, Arkansas State University, August, 1982, to May, 1983; Graduate Teaching Associate, Department of Management, Oklahoma State University, August, 1983, to May, 1986; Lecturer, Department of Management, Oklahoma State University, August, 1986, to December, 1986; Assistant Professor of Operations Management, University of Missouri-Kansas City, January, 1987, to present.

Professional Organizations:
The Institute of Management Science
Decision Sciences Institute
American Production and Inventory Control Society
American Society for Quality Control