Abstract

WARDONO, BAGAS. Algorithms for the multi-stage parallel machine problem with buffer constraints. (Under the direction of Yahya Fathi).

In this dissertation we study two problems in the context of scheduling N jobs on parallel machines in L successive stages. The primary objective in both problems is to minimize the makespan. In the first problem we assume that there are unlimited buffer capacities between stages. We refer to this problem as the flowshop with parallel machine (FSPM) problem. In the second problem, we assume that the buffer capacities are finite and given. We refer to this problem as the flowshop with parallel machines and limited buffer capacities (FSPM/b). Both problems are shown to be NP-hard in the strong sense. Thus, it is unlikely that either problem can be solved in polynomial time. A literature review shows that there has been little contribution in the open literature in this field, especially for the case of FSPM/b problem. In this study, we focus our attention on developing heuristic search procedures for solving both problems. We also develop appropriate

lower bounds for the corresponding makespan.

The heuristic procedures that we consider include the local improvement, the genetic algorithm, and the tabu search. In the context of these search procedures, we use two different solution representations, namely the *matrix representation* and the *vector representation*. The matrix representation consists of a pair of matrices that together provide a complete representation for the FSPM problem. This representation is relatively large and cumbersome, but it can represent any solution for the FSPM problem. The vector representation, on the other hand, is an abbreviated representation that limits itself to a select subset of the solutions of the FSPM problem. This representation consists of a single vector that specifies the starting order of the jobs at the first stage. Naturally, an associated procedure is required to construct a complete solution corresponding to a given vector, and the effectiveness of the search procedure strongly depends on the effectiveness of this constructive procedure. We propose two different constructive procedures in this context. These two procedures, along with the search strategies mentioned earlier, lead to several different search procedures for the FSPM problem.

We carry out an extensive computational study to evaluate these search procedures. We use two different data sets in this experiment. The processing times of the jobs in both data sets are randomly generated. We design the structure of the instances in the first data set in such a manner that the corresponding optimal makespan can be analytically determined. This optimal makespan is used

as a reference value for these instances. We adopt the structure of the instances in the second data set from the open literature. The optimal makespans for these instances are not known to us. We develop a set of lower bounds to be used as reference values for these instances.

The results of this computational study show that the proposed search procedures which employ vector representation perform reasonably well. In particular, we observe that one of these search procedures, namely the tabu search procedure, performs comparable with a previously reported tabu search procedure that employs an elaborate matrix representation. Yet, this procedure is much more flexible than its matrix representation counterpart and it can be easily extended (with appropriate modifications) to a search procedure for the FSPM/b problem. Such an extension would be extremely difficult, if at all possible, for the search procedure with matrix representation.

Thus, in the second part of the document we employ the vector representation to design a tabu search for the FSPM/b problem. A key step in this context is to devise an appropriate procedure to construct a complete schedule for the FSPM/b problem associated with a given vector. We propose two different procedures in this context, leading to two different tabu search procedures for the FSPM/b problem. We conduct another computational experiment to evaluate these two procedures. In this experiment we also construct two types of data sets in the same manner as those for the FSPM problem. In addition, we also experiment with a third

set of data which is available in the open literature. In order to measure the solution quality of the data set with unknown optimal solution, we use the set of lower bounds developed for the FSPM problem. From the experimental results, we demonstrate the characteristic of the proposed algorithms with various buffer configurations.

ALGORITHMS FOR THE MULTI-STAGE PARALLEL MACHINE PROBLEM WITH BUFFER CONSTRAINTS

by

BAGAS WARDONO

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

GRADUATE PROGRAM IN INDUSTRIAL ENGINEERING

Raleigh
2001

APPROVED BY

Chair of Advisory Committee

Biography

Bagas Wardono was born in September 1964 in Indonesia. He attended Institut Teknologi Sepuluh Nopember (ITS) in Surabaya, Indonesia, from August 1983 to September 1987. He received a bachelor degree in Mechanical Engineering as an honor student. Upon graduation, he worked as a mechanical engineer for a design and manufacturing company in Surabaya, Indonesia, where he was involved in various projects.

In August 1992, he entered Iowa State University in Ames, Iowa, to pursue a master's degree. He received his degree in Mechanical Engineering in 1994. In August 1997, he joined the Graduate Program in Industrial Engineering at North Carolina State University. He received his doctorate degree in 2001.

Acknowledgments

I would like to express my deepest appreciation to the faculty of the Department of the Industrial Engineering at North Carolina State University in advancing my education. Especially, I would like to thank Dr. Yahya Fathi for his invaluable guidance and sincere help for the completion of this dissertation. I also would like to express my gratitude to the members of the committee Dr. Salah E. Elmaghraby, Dr. Henry L.W. Nuttle and Dr. Russell E. King, for their encouragement and constructive suggestions.

Finally, I would like to thank my parents, my brother and sisters for their endless support during my study.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Flow shop scheduling with parallel machine (FSPM) to minimize makespan has received great attention in the literature. The problem deals with a set of jobs and a set of processing stages with a set of identical parallel machines at each stage. Each job consists of L operations that must be completed on the L stages and each job passes through the same stage routing sequence. At each processing stage, a job can be processed on any machine.

For $L = 1$ with one machine available, the problem is trivial. For a two-stage flowshop problem with one machine at each stage, Johnson algorithm [27] provides an optimal solution in the order of $O(n \log n)$. For $L \geq 3$ with one machine at each stage, Garey et al. [20] have shown that the problem is NP-Complete in the strong sense whether preemption is allowed or not.

A mathematical programming formulation is available for the L-stage FSPM

problem with independent setup times and infinite buffer sizes [24]. Several exact algorithms that have been designed and implemented for solving this problem include the branch-and-bound method [57]. However, computational experiments show that these algorithms become onerous as the number of jobs increases [18].

The problem becomes even more complex if there are only a limited number of buffer spaces available between successive stage. We refer to the resulting problem as the FSPM/b, where b is a vector representing the size of the buffers, as we discuss later.

In this study, we design and implement several heuristic procedures for solving the FSPM and FSPM/b problems. These procedures are based on various search techniques such as local improvement, tabu search, and genetic algorithm. Two types of solution representations are used and we study their effectiveness in the context of various search techniques. To evaluate the resulting algorithms, we randomly generate a set of problem instances (both with unlimited buffer capacities and with limited buffer capacities). The algorithms are then applied to these instances, and the results are evaluated on an empirical basis.

The primary contributions of this research can be summarized as follow.

1. Development of several search procedures for solving the FSPM problem which are based on a vector representation of the solution. This is in sharp contrast with previous work in this area which primarily uses a matrix representation of the solution.

2. Development of new and effective lower bounds for the FSPM problem.

3. An extensive computational study to evaluate the effectiveness of the proposed search procedures. In addition to using the existing test problems from the open literatures, in this experiment we also introduce a new set of test problems with known optimal solutions.

4. Development of two tabu search procedures for the FSPM/b problem and report on their performance via extensive computational studies.

1.1 Problem definition

To analyze the problem, we introduce several assumptions as follows.

- All jobs are assumed to be available at the beginning of the planning horizon.
- Each machine can only process one job at a time.
- The processing time for each job at each stage is known, and it is the same for all machines at that stage.
- Machines are available all the time and preemption is not allowed.
- Setup times are included in the processing times and independent of the job sequence.
- The buffer capacity between any two successive stages is given.

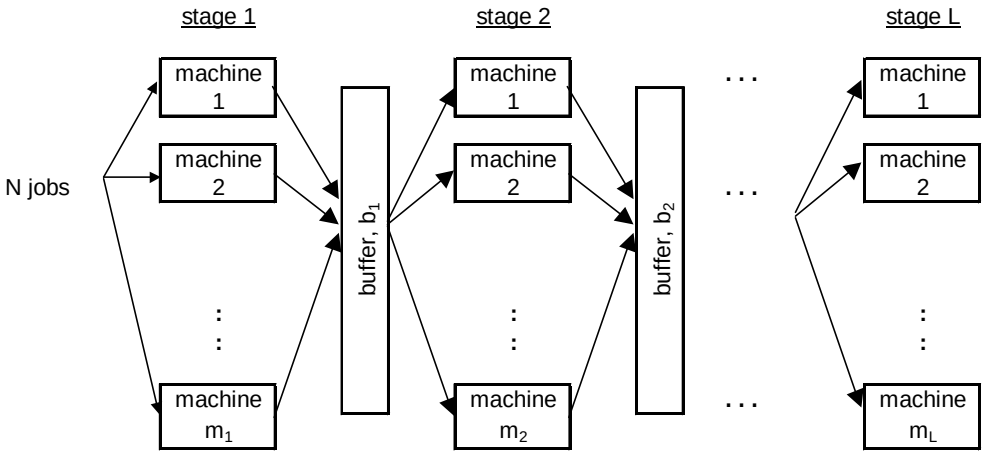


Figure 1.1: The schematic of the hybrid flowshop with buffers.

The flowshop system under consideration has the schematic as shown in Figure 1.1. The system consists of L processing stages, with m_ℓ identical parallel machines available at stage ℓ ($m_\ell \geq 1$), for $\ell = 1, ..., L$. The buffer between stages ℓ and $\ell+1$ is referred to as the ℓ^{th} buffer, and its capacity is denoted by b_ℓ, for $\ell = 1, ..., L-1$. The set of N jobs are denoted by $j = \{1,..., N\}$. Each job consists of L operations that must be completed on the L processing stages and each job passes through the same machine routing sequence. At each processing stage, a job can be processed on any machine. The corresponding processing time of job j at stage ℓ is given and denoted by $p_{j\ell}$, for $j = 1, ..., N$ and for $\ell = 1, ..., L$. Upon completion at stage ℓ, a job may be processed at the subsequent stage when a machine is available or it could reside in an empty buffer space following stage ℓ. In case when an empty buffer space is not available, the job must remain on the machine until either a

buffer space is available or a machine at the next stage becomes available. This job is then referred to as a *blocked job*, and the corresponding machine is referred to as a *blocked machine*.

At each stage, a job can only be processed by one machine and a machine can only process one job at a time. Each job j, $j \in \{1,...,N\}$, goes through the L operations according to a *schedule* that specifies the machine and the sequence assignment at each stage ℓ. To each given schedule, there is a *makespan*. The objective of the problem is to find a schedule that minimizes the makespan.

We distinguish between two versions of this problem. In the first version, we assume that all buffer capacities are infinite. We refer to this problem as the *Flow Shop with Parallel Machine Problem* or FSPM. This problem is also commonly referred to as hybrid flowshop, flexible flowshop, or multiprocessor flowshop. In the second version, we assume that the buffer capacities are finite and given by a vector b. We refer to this version of the problem as FSPM/b. There are some significant differences between these two problems. Thus, we treat these two problems separately.

1.2 Solution representation

Prior to any attempt at solving the problem, we must determine the appropriate format to represent a solution. In this study, we use two different formats. The first, which we refer to as the *matrix representation*, is a modification of the for-

mat proposed by Nowicki et al. [44]. This format is capable of representing any solution to the problem, but is somewhat cumbersome for algorithm development, particularly as it pertains to the problem with limited buffer size.

The second, which we refer to as the *vector representation*, is much simpler to work with. However, its scope is somewhat limited. It is the primary objective of this study to compare these two solution representations in the context of various search algorithms. We briefly describe these two solution formats here, and discuss their properties later.

1.2.1 Matrix representation

We represent a solution for the problem using two N by L matrices I and X as defined below.

$I_{j\ell} =$ the machine number to which job j is assigned at stage ℓ, for $j = 1, ..., N$ and for $\ell = 1, ..., L$.

$X_{j\ell} =$ the position of job j within the sequence of jobs that are assigned to machine $I_{j\ell}$ at stage ℓ for $j = 1, ..., N$ and for $\ell = 1, ..., L$.

Using this representation and together with the neighborhood structure that we define later, the matrix representation could accommodate all possible schedules.

For notational convenience, we also define $J_{i\ell}$ as the collection of jobs assigned to machine i at stage ℓ. It follows that

$J_{i\ell} = \{j \in (1, 2, ..., N) : I_{j\ell} = i\}$.

1.2.2 Vector representation

Using matrix representation as described above, our preliminary investigation with various search methods for solving FSPM problem shows that the resulting procedures can be ineffective. This is particularly true for the problem with limited buffer capacities. We believe this is mainly due to the fact that defining an appropriate neighborhood structure for this solution representation is relatively cumbersome and the resulting neighborhood size is inevitably too large and inefficient (i.e., it contains many solutions which are obviously inferior to the current solution, yet they need to be investigated in the context of the search procedure).

In an effort to avoid these inefficiencies, we define a different solution representation which we refer to as the *vector representation*. This representation is based on the point of view that in a good schedule, a job is likely to be processed at each stage as soon as it becomes available for processing at that stage (i.e., immediately after its completion time at the previous stage). Obviously, the processing time of each job at each stage also plays an important role. As such, we define a solution by a permutation vector of size N, which represents an ordering of the jobs 1 through N. This ordering is used to determine a schedule of the jobs at the first stage. The job allocation and ordering for each subsequent stage is then determined based on the order of completion time of the jobs at the previous stage

as well as the processing time for the job at the remaining stages. More specifically, for a given vector, the corresponding schedule is obtained in the following manner.

Step 1 - Allocate the jobs to the m_1 parallel machines at the first stage according to the First Available Machine (FAM) rule. (This rule is well known and discussed in the literature, see [32]).

Step 2 - Devise an appropriate procedure for scheduling the jobs at each subsequent stage based on the order of their completion time at the previous stage, the buffer availability, and their given processing times. We discuss several such procedures in subsequent chapters as appropriate.

1.3 Research objectives

In this research study, we have the following objectives.

1. Design and implement an appropriate procedure for the FSPM problem using the matrix representation and to test its effectiveness. In this context, we focus only on the local improvement technique.

2. Design and implement appropriate procedures for FSPM and FSPM/b problems using the vector representation and to test their effectiveness. The procedures include local improvement, tabu search, and genetic algorithm.

3. Carry out a computational experiment to evaluate these procedures and choose the most suitable procedure.

1.4 Organization of the dissertation

The organization of the remaining parts of this dissertation is as follow. Chapter 2 presents the literature review of flowshop and hybrid flowshop. In chapter 3, we present the local improvement procedure that we have developed for the FSPM problem using matrix representation. The discussion includes the method of obtaining the objective function, the neighborhood structure, and the local improvement procedure itself. Chapter 4 discusses the heuristic procedures that we have developed for solving the FSPM problem using vector representation. First, we present two procedures for generating a complete schedule for a given vector representation. We then discuss our design of a local improvement procedure, a genetic algorithm, and a tabu search procedure for this problem using vector representation. The computational experiments and results using these algorithms are presented in chapter 5. Here, we present the type of problem instances used for the experiment and the method of obtaining the lower bound to the optimal solution. We also present the results of a study to investigate the impact of various algorithmic features (such as different neighborhood structures and different methods for constructing a complete schedule) for a given vector representation. In the last part of this chapter, we present the methods of evaluating the performance of the algorithms in terms of the solution quality, and the experimental results corresponding to them. In chapter 6, we present the procedures that we developed for the FSPM/b problem. In this chapter, we also present the method of deter-

mining the feasibility of a solution. The computational results corresponding to this problem are shown in chapter 7. Finally, in chapter 8 we present a summary of the work that we have completed and the suggestions for future research.

Chapter 2

Literature Review

There is a large number of articles in the open literatures that discuss the flow shop scheduling problem with various features. The flowshop scheduling problem on m machines in series to minimize makespan (C_{max}) is denoted by $Fm/-/C_{max}$. The famous Johnson algorithm [27] provides an optimal solution to the two-machine flowshop problem. The problem assumes an unlimited buffer storage between the two machines. The algorithm follows immediately from the rule: *Job i precedes job j if* $\min\{p_{i1}, p_{j2}\} \leq \min\{p_{j1}, p_{i2}\}$, where $p_{j\ell}$ is the processing time of job j at stage ℓ. An extension to this problem with three stages has been shown to be NP-hard [20]. Due to the increased difficulty as the number of stages and the number of parallel machines at each stage increase, several heuristic methods, including constructive heuristics and local search techniques, have been developed. In this chapter, we present a brief review of these methods. In sections 2.1 and 2.2, we

focus on the review for the multistage single machine problem. In section 2.3, we discuss the literature for the multi stage with parallel machine problem.

2.1 Constructive heuristics for $Fm/-/C_{max}$ problem

The constructive heuristics generate a schedule from scratch. The main advantage of the constructive heuristics is that they are usually simple and they do not require significant CPU time to secure a complete schedule. Nawaz, Enscore, and Ham [39] developed a constructive heuristic for the flowshop scheduling with one machine at each stage. Their algorithm (NEH) builds the final sequence by adding one job at a time. NEH algorithm is based on the assumption that a job with high total processing time should have a higher priority than a job with lower total processing time. Widmer and Hertz [65] suggest building the algorithm by using the analogy to the travelling salesman problem. A distance matrix between two jobs is generated first; next, a sequence is developed using the insertion method.

Some other constructive heuristic algorithms ([7], [13], [23], [47]) were developed based on the ideas of Johnson's algorithm. Campbel et al. [7] developed their algorithm by introducing $m-1$ artificial problems. In artificial problem k, the problem is transformed into two machines with the processing times on machine one equal to the sum of the processing time on the first k machines and the

processing times on machine two equal to the sum of the processing time on the last k machines. The two-machine problem is then solved using Johnson algorithm and the resulting sequence is used for the m-machine problem. This procedure is repeated for all values of $k = 1, ..., m-1$, and the best resulting makespan among them is reported. Among constructive heuristic algorithms for $Fm/-/C_{max}$ problem, it has been shown through numerical experiment that NEH is superior to others in terms of the solution quality ([61], [62]).

Flowshop problem with zero buffer capacities between stages is denoted by $Fm/block/C_{max}$. For this type of problem, Levner [34] proposed an algorithm which is based on the branch and bound method. This algorithm finds an optimal solution for the problem. For a similar problem, Reddi and Ramamoorthy [51] have shown that the problem is equivalent to the travelling salesman problem.

For flowshop problem with the inclusion of release time, the reader is referred to [5], [8], and [60].

2.2 Local search methods for $Fm/-/C_{max}$ problem

The local search techniques start with a complete sequence of schedule. Many algorithms use random sequence generation to construct the initial schedule and some of them use a known constructive heuristic for this purpose. The local search

heuristic procedures typically provide a scheme to obtain a new solution in the neighborhood of the current solution. These procedures include local improvement, simulated annealing, and tabu search.

Ogbu and Smith [45] use simulated annealing method to solve the $Fm/-/C_{max}$ problem. They use random generation as well as two well-known heuristics ([13], [47]) to construct the initial solution. They conclude that starting the simulated annealing algorithm using good heuristics provides no discernible advantage compared with a random initialization. They also conclude that the average relative error of their algorithm to the optimal solution is less than 2%.

Similar with Ogbu and Smith [45], Osman and Potts [46] also implement simulated annealing method to solve the $Fm/-/C_{max}$ problem. They compare their computational results with the results of heuristic NEH. The results suggest that the simulated annealing method gives better results than the NEH heuristic. For further comparison, they consider two descent methods, i.e., the descent method that uses NEH as the starting sequence and the descent method that uses the best of $m+1$ sequences generated from the constructive methods of Palmer [47], Campbel, Dudek, and Smith [7], and Dannenbring [13]. Of these two descent methods, the latter requires much more computation time due to inferior starting point, although the solution qualities are comparable. Therefore, Ogbu and Smith compare their results with the first descent method. The results show that the simulated annealing method gives an average relative percentage deviation from

the best solution of 0.49% as compared with 1.15% given by NEH descent method. (Notice that a large number of simulated annealing iterations was performed to obtain the best objective value).

Another simulated annealing method was introduced by Zegordi et al. [67]. They define the criterion of the desirability of moving a job forward or backward. The index of each job to be scheduled is calculated based on this criterion. Then, pairwise exchanges are selected based on the index. To evaluate the results, Zegordi et al. compared their results with the NEH [39] and the Campbel, Dudek, and Smith [7] algorithms. The comparisons were based on the frequency criterion (number of cases in which an algorithm gives better objective function values than others) and the deviation criterion (the mean relative percentage deviation of the objective function values given by two algorithms). The results show that the simulated annealing method by Zegordi et al. gives higher quality solution than the NEH or CDS heuristic. Comparison with another simulated annealing method by Osman and Potts [46] shows that they both give a relatively similar solution quality. However, as the problem size gets larger, the algorithm by Zegordi et al. is considerably superior in terms of the computational effort. Another local search method used to solve $Fm/-/C_{max}$ problem is the tabu search method as described in reference [4] and [43].

A variation of the $Fm/-/C_{max}$ problem is the inclusion of limited buffer capacity between any two machines. Nowicki [42] uses the tabu search algorithm

to solve the problem. He assumes that between each successive pair of machines $i-1$ and i, there exists FIFO rule. Each job must go through a buffer on its route between two machines. In case no buffer is available, the job must remain in the current machine until a buffer becomes available. For comparison, Nowicki compared his computational results with the results of the generalization of the NEH algorithm as described by Leisten [33], SA-fast and SA. (SA-fast is a mutation of simulated annealing algorithm to avoid excessive running time for large instances as introduced by Das et al. [14]). The comparisons show that the tabu search method gives better results as compared with the generalized NEH, SA, and SA-fast in terms of the solution quality. In terms of the significance of the buffer size, Nowicki concluded that the best permutation found for the case with infinite buffer capacity is bad for the case of finite buffer capacity when the size of buffer is relatively small.

In addition to the limited buffer capacity, Norman [41] includes sequence dependent setup times to his model. This problem takes into account the common issue that arises in chemical processes where the setup time of job j on machine m is dependent on the job that has been executed on machine m immediately before. Norman developed both the constructive heuristic combined with the descent method and the tabu search. The computational results were then compared with each other and with NEH algorithm (with and without descent method). The results show that the tabu search found solutions that are consistently better

than those found using constructive heuristic method combined descent method. The tabu search remains better for various combinations of numbers of jobs and machines, buffer configuration, and setup times.

2.3 Previous contributions on FSPM problem

The discussion of the various heuristics above is based on having only one machine available at each stage. If we allow several machines in parallel at each stage, the resulting problem is known as the flowshop problem with parallel machines (FSPM), or hybrid flowshop, problem. Several studies have been presented for this problem as discussed below.

In his paper, Chen [9] presented an approach to the two-stage hybrid flowshop problem where the number of machines at each stage are identical. The constructive heuristic approach by Sundararaghavan et al. [59] suggests a solution to a two-parallel flowshop problems where the number of machines at each flowshop is equal to two. They assume that the set of machines in one flowshop is faster than the set of machines in the other flowshop. Once a job is assigned to a flowshop, it cannot be switched to the other flowshop in the next stage.

Lee and Vairaktarakis [32] developed a constructive heuristic for the two-stage hybrid flowshop problem and then they developed a general constructive heuristic applied to the L-stage hybrid flowshop problem. For the two-stage hybrid flowshop problem, Lee and Vairaktarakis employ the Johnson algorithm [27] to generate the

initial job sequence followed by the first available machine rule (FAM) and the last busy machine rule (LBM) to generate the final sequence. For the L-stage hybrid flowshop, they implement their algorithm for the two-stage hybrid flowshop for each pair of stages (assuming that the number of stages L is even, otherwise a dummy stage with zero machine is introduced).

For the 2-stage FSPM problem, Haouari and M'hallah [25] developed a constructive heuristic. Then, they improved this heuristic using simulated annealing and tabu search procedures. In implementing the local search procedure, they use a permutation vector of the N-jobs as the solution representation.

A tabu search approach for the L-stage hybrid flowshop is presented by Nowicki et al. [44]. The objective is to minimize the makespan. In this approach, a schedule is defined by the collection of pairs $(I_{j\ell}, C_{j\ell})$, where $I_{j\ell}$ is the machine number to which job j is assigned at stage ℓ and $C_{j\ell}$ is defined as the completion time of job j at stage ℓ, for $j = 1, ..., N$ and for $\ell = 1, ..., L$. The neighborhood structure that was implemented is the insertion method. This structure is defined using a vector $v = (\ell, a, x, b, y)$ designating the removal of a job in the x^{th} position on machine a at stage ℓ and its insertion in the y^{th} position on machine b at the same stage. Using the notion of *useless moves*, they establish some rules to eliminate some moves that are known a priori to yield no improvement on the makespan. Thus the number of neighbors that has to be evaluated can be reduced. To start the search, the authors use the processing order rule $DK1$ in [18] as the initial solution.

To evaluate the performance of their algorithm, the authors use two types of quality measures. The first scheme is to measure the relative improvement of their algorithm with respect to the reference makespan. Here, the authors compared the resulting makespan with reference makespan which is determined as the best makespan obtained using several known constructive algorithms ($DK1$, $DK2$, $DK3$ [18], W [66], S [56], and HS [26]). The second scheme is to measure the relative distance of the resulting makespan with respect to the lower bound on the minimum makespan. For this purpose, they use the lower bound to the problem which is obtained by introducing job preemption and the relaxation of machine capacities [31]. Nowicki et al. [44] also developed a local improvement procedure for this problem, but the results are not as good as those obtained using the tabu search.

Recently, Negenman [40] developed several local search algorithms using tabu search procedure, simulated annealing, and variable-depth search procedures. With these algorithms, Negenman performs an experiment with several different neighborhood structures including the neighborhood structure introduced by Nowicki et al. [44]. To measure the solution quality, Negenman uses the set of lower bounds developed by Vandevelde [64]. The results show that tabu search and variable-depth search with the neighborhood structure by Nowicki et al. are superior to the other algorithms.

For the problem with buffer constraint, i.e., FSPM/b problem, Wittrock [66] has

developed a constructive algorithm to solve six instances from the real production line. Each instance consists of three stages with the number of machines $m_1 = m_2 = 2$ and $m_3 = 3$. To measure the solution quality, Wittrock proposed a lower bound which is defined as the maximum of the average workload among machines at each stage. Later, Sawik [56] developed a different constructive algorithm to solve the six instances introduced by Wittrock. The result shows that in five out of six instances, Sawik's algorithm gives better results than the algorithm by Wittrock.

Chapter 3

FSPM - Algorithm Based on Matrix Representation

In this chapter, we describe the search procedure for the FSPM problem with unlimited buffer sizes based on the matrix representation. As we described earlier, in this representation a solution consists of two matrices I and X. In section 3.1, we describe a procedure to determine the makespan for a given solution (I, X). In section 3.2, we define the neighborhood structure based on this solution representation. In section 3.3, we propose the local improvement procedure.

3.1 Determining the makespan

For a given solution, i.e., I and X matrices, we determine the corresponding makespan as follow.

Let $n_{i\ell}$ be the number of jobs assigned to machine i at stage ℓ, i.e., $n_{i\ell}=|J_{i\ell}|$ (recall that $J_{i\ell}$ is the set of jobs assigned to machine i at stage ℓ). Let $\pi_{i\ell}$ be a vector of size $n_{i\ell}$ that represents the sequence of jobs assigned to machine i at stage ℓ (i.e., $\pi_{i\ell}$ is a permutation vector of the element of $J_{i\ell}$). It follows that

$$\pi_{i\ell}(k) = \{j \in J_{i\ell} : X_{j\ell} = k\} \quad , k = 1, ..., n_{i\ell};$$
$$, i = 1, ..., m_\ell;$$
$$, \ell = 1, ..., L,$$

(i.e., $\pi_{i\ell}(k)$ is the job that is processed at the k^{th} position on machine i at stage ℓ).

Let $C_{j\ell}$ be a matrix of size N by L representing the completion time of job j at stage ℓ. Values of $C_{j\ell}$ for all j and ℓ can be obtained in a recursive manner starting with $\ell = 1$ as follows. For stage 1, we have

$$C_{\pi_{i1}(1),1} = p_{\pi_{i1}(1),1} \qquad , \text{for } i = 1, ..., m_1$$

and

$$C_{\pi_{i1}(k),1} = C_{\pi_{i1}(k-1),1} + p_{\pi_{i1}(k),1} \quad , \text{for } i = 1, ..., m_1$$
$$, \text{for } k = 2, ..., n_{i\ell}.$$

For the subsequent stages $\ell = 2, ..., L$, we have

$$C_{\pi_{i\ell}(1),\ell} = C_{\pi_{i\ell}(1),\ell-1} + p_{\pi_{i\ell}(1),\ell} \qquad , \text{for } i = 1, ..., m_\ell$$

and

$$C_{\pi_{i\ell}(k),\ell} = \max\{C_{\pi_{i\ell}(k),\ell-1}, C_{\pi_{i\ell}(k-1),\ell}\} + p_{\pi_{i\ell}(k),\ell} \quad , \text{for } i = 1, ..., m_\ell$$
$$, \text{for } k = 2, ..., n_{i\ell}.$$

Upon evaluation of the completion time matrix, $C_{j\ell}$, the makespan (C_{max}) of the completed schedule can be determined using $C_{max} = \max_j \{C_{jL}\}$.

3.2 Neighborhood structure

Given a solution (I, X), we define its neighborhood $N(I, X)$ as follows. A solution (I', X') is said to be in the neighborhood of (I, X) if it is obtained from (I, X) by changing the processing position of one job at one stage. More specifically, $(I', X') \in N(I, X)$ if it is obtained from (I, X) by removing a job from its current position (say the job at the x^{th} position on machine a at stage ℓ), and inserting it at a different position (say at the y^{th} position on machine b at the same stage). As such, for a given solution (I, X), a solution in its neighborhood can be uniquely specified by a vector $v = (\ell, a, x, b, y)$. It follows that each solution (I, X) has $NL(N+m_\ell-2)$ neighborhood points. To make the search procedure more effective, we can reduce the size of this neighborhood by limiting the set of jobs that are considered for moving. For this purpose, we adopt the notion of *critical path* as introduced by Nowicki et al. [44].

We define a job to be critical in a given solution if its processing time contributes directly to the makespan. Therefore, any delay to the critical job will cause a delay in the makespan. The sequence of jobs formed by the critical jobs is called a *critical path*, which is the longest path from the first stage to the last stage. We observe that implementing the perturbation scheme mentioned above by moving

a non-critical job from its current position cannot possibly improve the objective function. Therefore, the insertion method is implemented by considering to move only the critical jobs. Since the critical path is not necessarily unique, we choose arbitrarily a critical path in case there are more than one such critical paths.

3.3 Local Improvement procedure

The local improvement procedure (LI) is started by generating an initial solution (I, X). The initial solution is generated randomly as described later in this section. We refer to this initial solution as the "current solution". The next step of the procedure is to find a solution in the neighborhood of the current solution with a better makespan. If such a solution is found, we set it to become the "current solution". We continue this process until no solution with a better makespan can be found in the neighborhood of the current solution, at which time we terminate the procedure. The schematic of the local improvement procedure is shown in Figure 3.1.

The main feature of the local improvement procedure is that it moves only to a neighboring solution that improves the current solution. Thus, it is possible that this procedure would converge to a local optimum which is not a global optimum. To increase the likelihood of finding a global optimal solution, we repeat the algorithm using several different starting solutions.

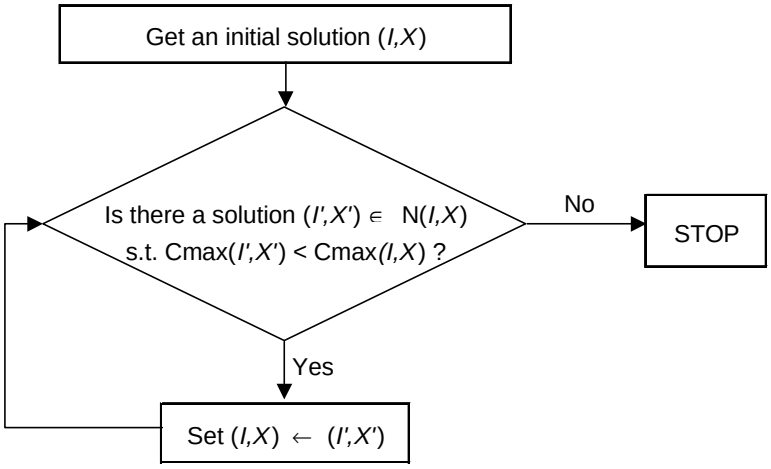


Figure 3.1: The schematic of the local improvement procedure.

3.3.1 Initial solution

For the purpose of our study, we generate the initial solution randomly by generating a random list of jobs 1 through N. Starting from the first job j in this list, we assign this job to the first available machine at stage $\ell = 1$. We then record the completion time ($C_{j\ell}$) for this job, the machine that processes this job ($I_{j\ell}$), and the order of this job in the machine ($X_{j\ell}$). We continue this process until all jobs are assigned at stage 1. For the subsequent stages ($\ell > 1$), we implement the same procedure except that our selection of a job to be scheduled is based on the earliest completion time at the previous stage. The makespan (C_{max}) of the completed schedule generated using the initial solution (I, X) can be determined using $C_{max} = \max_j \{C_{j,L}\}$.

3.3.2 Search strategy

In a local improvement procedure, there are two search strategies that are commonly implemented, i.e., *first-improvement* and *steepest descent*. In the first-improvement strategy, we move to the first solution encountered that gives better makespan than the makespan of the current solution. On the other hand, the steepest descent method evaluates the entire neighborhood of the current solution and a move is determined by selecting the best neighbor. It is obvious that the computational effort at each iteration for the steepest descent strategy is higher than that for the first-improvement strategy. But the number of iterations is expected to be lower. After a preliminary computational study with multiple restart for each strategy, we observed that the first-improvement strategy performs better than the steepest descent strategy in the context of this implementation. In all subsequent studies, we use the first-improvement strategy.

Chapter 4

FSPM - Algorithms Based on Vector Representation

As mentioned earlier, the matrix representation allows us to uniquely represent any solution to the FSPM problem, and this is an advantage for this solution representation. But, in a limited computational study that we performed, we observed that the local search procedure that we devised based on this solution representation is somewhat inefficient. We believe this is due to the fact that the resulting neighborhoods are too large. Many of the neighborhood points considered are actually useless and need not be considered.

For example, consider job 8 as shown in Figure 4.1. Since job 8 at the second stage is part of the critical path, job 8 could be selected for the next move. Using our neighborhood definition for the matrix representation, at stage 2, job 8 could

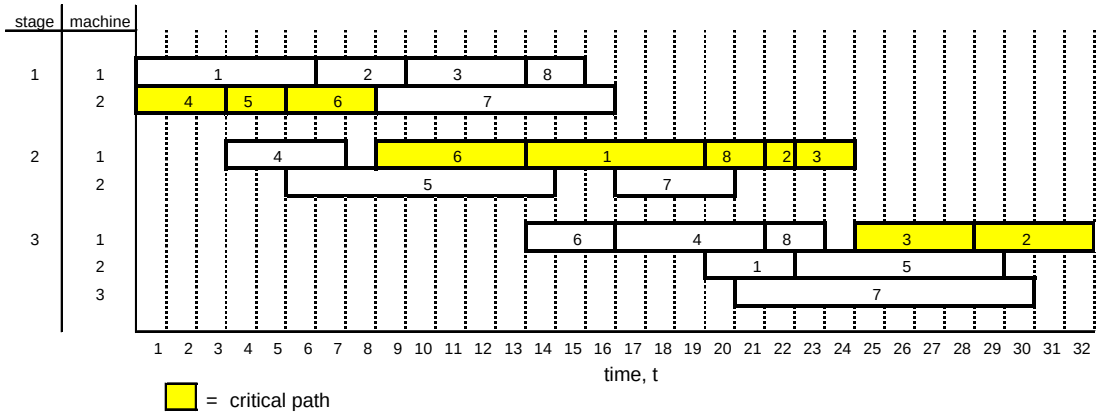


Figure 4.1: A sample of a feasible schedule with unlimited buffer sizes.

be inserted in five possible positions on machine 1 and in three possible positions on machine 2. However, one can observe that inserting job 8 in the first position on machine 1 at stage 2, as defined by vector $v = (2, 1, 4, 1, 1)$, causes job 1, 2, 3, 4, and 6 to reside in buffer b_1 upon completion at stage 1. This is due to the fact that at stage 1, job 8 is processed last. Therefore, this move will not improve the solution. Evaluating such moves can be very time consuming, and it results in an inefficient search procedure.

In this chapter, we present an alternative representation and a corresponding neighborhood structure that helps to avoid many such useless moves. Thus, it has the potential to improve the efficiency of the search procedure. In this approach, we use a permutation vector S of size N to represent a solution. This vector S represents the order in which the jobs are considered at the first stage. Subsequently, we devise appropriate procedures to uniquely construct a complete

schedule using this permutation vector and determine its corresponding makespan. This representation is much simpler (and we believe it is more efficient) than the matrix representation in the context of various search procedures. However, it is somewhat more limited, i.e., not all possible schedules can be considered using this representation. For example, consider job 8 which is processed last at stage 1 (as shown in Figure 4.1). Using matrix representation, we are able to represent a schedule such that job 8 is processed first at stage 2. On the other hand, using the procedure to construct a complete schedule devised for the vector representation, it is likely that job 8 will be processed last on a machine at stage 2. Therefore, we might not be able to represent a schedule using vector representation such that job 8 is processed first at the second stage.

In this chapter, we propose several search algorithms for the FSPM problem with unlimited buffer capacities based on vector representation. Our objectives in proposing these algorithms are as follows.

- To compare these algorithms with that discussed in chapter 3 in terms of the quality of solutions.
- To use these algorithms as a stepping stone to design algorithms for the more complex problem of FSPM/b.

In section 4.1, we discuss the subject of objective function evaluation, and describe several procedures for transforming a vector solution representation into a complete solution. In section 4.2, we propose a local improvement procedure.

Here, we also discuss the various neighborhood structures that we used for the local improvement procedure. In section 4.3, we propose a genetic algorithm for this problem based on vector representation, and finally, a tabu search algorithm is presented in section 4.4.

4.1 Constructing a complete schedule

In this section, we propose two procedures for constructing a complete schedule corresponding to a given permutation vector S and determining its associated makespan. These procedures are based on various constructive heuristic methods for the FSPM problem. We discuss these procedures in the following subsections.

4.1.1 Procedure $Z1$ - Using release dates and tails

In this procedure, given a permutation vector S, we construct its corresponding schedule in two steps. The first step is to implement the selected rule to sequence the jobs in the first stage given the solution vector S. The second step is to devise an appropriate procedure for scheduling the jobs at each subsequent stage based on the order of their completion time at the previous stage.

For the first step, we implement the *first available machine* (FAM) rule to schedule the jobs, starting from the job in the first position in S. This rule designates that the next job to be scheduled is assigned to the first available machine. This machine is defined as the earliest available machine.

In the second step, the method that we implement at the subsequent stages is inspired by the method developed by J. Carlier [8] for the 1-stage parallel machine problem with m identical machines with *release dates* and *tails*. We use a similar approach at each of the subsequent stages (after stage 1) in our multi-stage problem. In our approach here, we consider the completion time of a job at stage $\ell - 1$ ($\ell > 1$) to be its "release time" at stage ℓ and the sum of processing time of the job from stage ℓ to stage L to be its "tail". A similar method was also used by Haouari and M'hallah [25] to solve the 2-stage FSPM problem.

The description of this procedure which we refer to as procedure $Z1$ is as follow.

Procedure: $Z1$

For each stage $\ell = 2, ..., L$, perform the following procedures:

Step 1 - Initialize:

$$\begin{aligned}
q_j &= \sum_{k=\ell}^{L} p_{jk}, \text{ for } j = 1, ..., N && \text{(the remaining processing time of job } j\text{)} \\
u_m &= 0, \text{ for } m = 1, ..., m_\ell && \text{(availability of machine } m\text{)} \\
w_m &= 0, \text{ for } m = 1, ..., m_\ell && \text{(number of jobs processed on machine } m\text{)} \\
R &= \{1, ..., N\} && \text{(the set of unscheduled jobs at stage } \ell\text{)}
\end{aligned}$$

Step 2 - Let

$$\begin{aligned}
dummy &= \min_{j \in R} \{C_{j,\ell-1}\} && \text{(earliest job completion time)} \\
u_{k_o} &= \min_{m} \{u_m\}, \text{ for } m = 1, ..., m_\ell && \text{(earliest machine availability)}
\end{aligned}$$

k_o represents the first available machine (break all ties arbitrarily).

Step 3 - If $u_{k_o} < dummy$ then set $u_{k_o} = dummy$.

Step 4 - Let,

$$q_{max} = \max_{j \in R} \{q_j : C_{j,\ell-1} \leq u_{k_o}\}$$

$$j_o = \arg\max_{j \in R} \{p_{j\ell} : C_{j,\ell-1} \leq u_{k_o} \text{ and } q_j = q_{max}\}$$

j_o is the index of job that we select to assign in this step.

Step 5 - Set,

$$C_{j_o,\ell} = u_{k_o} + p_{j_o,\ell} \quad \text{(record the completion time of job } j_o\text{)}$$

$$X_{j_o,\ell} = w_{k_o} + 1 \quad \text{(record the order of job } j_o \text{ in machine } k_o\text{)}$$

$$I_{j_o,\ell} = k_o \quad \text{(record the machine that processes job } j_o\text{)}$$

$$u_{k_o} = C_{j_o,\ell} \quad \text{(update the availability of machine } k_o\text{)}$$

$$w_{k_o} = w_{k_o} + 1 \quad \text{(update } w_{k_o}\text{)}$$

Step 6 - Set $R \leftarrow R \setminus \{j_o\}$. If $R \neq 0$, go to step 2; otherwise, terminate the procedure for stage ℓ.

Following is a brief explanation of each step.

Step 1 - In this step, we define the remaining processing time of job j as the sum of the processing time of job j from stage ℓ to the last stage L. Then, we initialize the availability of machine m (u_m) and the number of jobs processed in machine m (w_m). We also define R as the set of unscheduled jobs at stage ℓ.

Step 2 and 3 - We determine the earliest completion time of job j, $j \in R$, at the previous stage. We then compare this value with the finishing time of the first available machine (u_{k_o}). Following the flowshop rule that a job cannot be started earlier than its completion time in the previous stage, we set $u_{k_o} = dummy$ whenever u_{k_o} is less than $dummy$.

Step 4 - We choose job j_o to be scheduled next. This job is indicated to be the job which has the largest remaining processing time, the largest processing time at the current stage, and its completion time is not greater than u_{k_o}.

Step 5 - Then, we continue with updating the machine availability (u_{k_o}) and recording the machine k_o that processes job j_o, the order of job j_o in machine k_o, and its completion time.

Step 6 - Upon completion with job j_o, we remove job j_o from R and proceed with the next job in R, if $R \neq 0$. Otherwise, we proceed with the next stage until stage L is scheduled.

The makespan (C_{max}) of the completed schedule can be determined using $C_{max} = \max_j \{C_{jL}\}$.

In step 5, we record the machine that processes job j_o ($I_{j_o,\ell}$) and the processing order of job j_o on machine k_o ($X_{j_o,\ell}$). Using this information, we could generate the complete schedule (I, X) corresponding to a given initial permutation vector S. Furthermore, the set (I, X) for a given initial permutation vector S can be used to

determine the corresponding makespan of the L-stage problem with limited buffer size (FSPM/b) as discussed later in chapter 6.

4.1.2 Procedure $Z2$ - Using the variation of FAM rule

In this procedure, similar to procedure $Z1$, we construct the schedule corresponding to a given initial permutation vector S in two steps. The first step is to implement the FAM rule to schedule the jobs at the first stage starting from the job in the first position in S. At the subsequent stages, we implement a variation of the FAM rule to schedule the jobs. A job whose completion time at the previous stage ($C_{j,\ell-1}$) is less than or equal to the time of the first available machine (u_{k_o}) is scheduled first. In case that several jobs satisfy this property, we choose the job with the longest processing time at the current stage.

In order to implement the variation of the FAM method at stage $\ell > 1$, we simply modify procedure $Z1$ above by removing the evaluation of q_j (the remaining processing time of job j) from step 1, and replace step 4 with the following.

Step 4 - Let,

$$j_o \quad = \quad \arg\max_{j \in R} \{p_{j\ell} : C_{j,\ell-1} \leq u_{k_o}\}$$

where j_o is the job that we select to assign in this step.

In a separate study, we conducted an experiment by implementing the pure FAM rule at all stages. The experiment was performed on several instances with

identical number of machines at all stages. The results showed that the FAM rule is inferior as compared with procedure $Z1$ in the context of tabu search. Therefore, we do not include the FAM rule in the context of FSPM problem as a part of our computational experiment.

4.2 Local improvement procedure

The local improvement procedure (LI) that we implement here is started by generating an initial solution S (i.e., a permutation vector). This initial permutation vector S is generated randomly. Upon evaluating the completion time matrix, C, corresponding to this solution (using the procedures as described in section 4.1), we evaluate its neighbors. A move to the selected neighbor is based on the first-improvement policy. When we encounter a neighbor that gives a better makespan, we move to this neighbor and repeat the process. The local improvement procedure stops when none of the neighbors has a better makespan and the current solution is a local optimum. To increase the chance of finding the global optimum, the algorithm is repeated using different starting points. The neighborhood structure that we used for the local improvement procedure is described below.

Given a vector (solution) S, we define its neighborhood $N(S)$ as follows. A solution S' is said to be in the neighborhood of S if it is obtained from S by changing the position of one job. Specifically, $S' \in N(S)$ if it is obtained from S by removing a job from its current position (say the job in the x^{th} position in S)

and inserting it at a different position (say at the y^{th} position in S with $y \neq x$), and shifting the position of all jobs between x and y by one unit accordingly. Therefore, a neighbor S' of S can be identified by a vector of size two, $v = (x, y)$. We refer to this neighborhood structure as *the general insertion scheme*, since it is defined for all possible values of x and y. It follows that each solution S has $N(N-1)$ neighborhood points.

In addition to this scheme, we also consider the insertion scheme where the removed job is inserted only into the first position (or the last position) in S. Using the last two schemes, the neighborhood size is $(N-1)$. For our future reference, we refer to these neighborhood structures as G, F, and L, where G refers to general insertion scheme, F refers to the insertion of the removed job to first position, and L refers to the insertion of the removed job to the last position.

4.3 Genetic Algorithm

Genetic algorithm (GA) is one of the various optimization techniques that have been implemented to solve combinatorial optimization. GA can be viewed as a search method where a better population of solutions is produced within each iteration. In addition, genetic algorithm uses stochastic search techniques in which the next moves are determined based on random sampling.

The main difference between GA and other search methods is the number of solutions maintained throughout the search. Most other search algorithms main-

tain one solution throughout the search, while GA maintains a set of solutions during the search for a better solution. Recently, interest in GA has tremendously increased. This is mainly due to the following:

- GA does not make assumptions about the linearity, convexity, or the differentiability of the objective function. (This property is shared by most other search methods as well).
- GA can be employed to maintain the balance of exploration and exploitation of the search space.
- In other search methods, a minor modification requires substantial changes. On the other hand, GA is generally easy to adopt to model variations of the original problem.
- GA parallels the evolutionary process and this is a promising direction in optimization [37].

In a particular problem, there are six fundamental issues related to GA implementation, namely chromosome representation, initialization of the population, objective function evaluation, selection mechanism, genetic operators, and stopping criterion. These issues are discussed in the following subsections.

4.3.1 Description of the algorithm

In our implementation of GA for the FSPM problem, each solution or *individual* in the population is represented by a vector of size N, which is commonly called a *chromosome*. The chromosome representation describes the individuals in the population maintained by GA. Thus, with regards to our FSPM problem, a chromosome representation refers to the permutation vector S.

GA is initialized by generating a population of solutions, which is typically done randomly. The objective function value corresponding to each solution is then evaluated to determine its fitness value. A probabilistic rule is performed to produce the next generation. This probabilistic rule is applied such that the better individuals have an increased chance of being selected. With this probabilistic rule, an individual can be selected more than once for the next step. These parents then reproduce a new population using the genetic operators. The GA moves from generation to generation in this manner until the stopping criterion is met. The GA procedure that we implement to solve the FSPM problem is described below.

Procedure: GA

1. Set the number of iterations, $iter = 0$

2. Generate a set of chromosomes $Pop(i)$, for $i = 1, ..., pop\_size$, where pop_size is the *population size*.

3. Construct a complete solution corresponding to each chromosome using pro-

cedure $Z1$ (or procedure $Z2$), and evaluate its corresponding objective function value, C_{max}. Let S^* be the solution with the lowest objective function value.

4. Set $iter = iter + 1$

5. Implement the selection rule to generate a new population $Pop'(i)$, for $i = 1, ..., pop\_size$, and let $Pop(i) \leftarrow Pop'(i)$.

6. Randomly select a set of individuals to perform the crossover operator.

7. Randomly select a set of individuals to perform the mutation operator.

8. Construct a complete solution corresponding to each new individual using either procedure $Z1$ (or procedure $Z2$), and evaluate its corresponding objective function value C_{max}.

9. Let S' be the best solution of the new population. If $C_{max}(S') < C_{max}(S^*)$, set $S^* \leftarrow S'$.

10. If the stopping criterion is not satisfied, go to step 4; otherwise, terminate the procedure and return S^* as the best solution obtained.

The population $Pop(i)$ consists of the set of permutation vectors S. In step 2, we generate the initial population by randomly generating a set of pop_size permutation vectors each of size N. To determine the objective function value of

each individual, we apply the procedures as described in section 4.1 (i.e., procedure $Z1$ or $Z2$).

The generation is repeated until the stopping criterion is met. As for the stopping criterion, we terminate the algorithm if there is no improvement to the best solution obtained after a certain number of generations ($Count$). In addition, we also limit the maximum number of generations ($iter\_max$).

4.3.2 Selection mechanism

Once we have evaluated the current population (step 3), a new population needs to be generated from the previous population (step 5). The selection strategy plays an important role in genetic algorithm. The purpose of the selection rule is to generate a new population which leans towards better individuals (note that the selected individuals are not necessarily distinct).

There are two main issues in performing such selection: population diversity and selective pressure [37]. Generally speaking, measures that result in an increase in the population diversity tend to decrease the selective pressure. As a result, the search can be ineffective. On the other hand, increasing selective pressure reduces the population diversity. Hence, this may cause a premature convergence of the search. Therefore, it is important to balance these two issues.

There are several methods that can be employed for selecting individuals. These methods include the roulette wheel method, scaling techniques, and the ranking

method. The selection rule that we implement here is the *ranking method*, i.e., the selection mechanism is based on the rank of each individual in the current population with respect to their objective function values. Thus, the main consideration is the relative fitness of each individual compared to other individuals in the population. In addition, we adopt the method as suggested by Michalewics [37]. In this method, individuals in the population are ranked from the best to the worst according to their objective function values. Then, a probability of selection (Pr) is assigned to each individual based on a nonlinear function with a user defined parameter q as follow.

$$Pr(rank) = q - (rank - 1) \times r \qquad \text{, for } rank = 1, ..., pop\_size$$

The highest rank is indicated by $rank = 1$ while the lowest rank is indicated by $rank = pop\_size$. Using the fact that $\sum_{rank=1}^{pop\_size} Pr(rank) = 1$, we can solve for q. The result is $q = \frac{r(pop\_size-1)}{2} + \frac{1}{pop\_size}$. Notice that if $r = 0$, then there is no selective pressure. Each individual has the same probability ($q = \frac{1}{pop\_size}$) of being selected. If we define $q-(pop\_size-1)\times r = 0$, this provides the maximum selective pressure with $q = \frac{2}{pop\_size}$ and $r = \frac{2}{pop\_size(pop\_size-1)}$. Michalewics [37] suggests that the selective pressure is controlled by varying q between $1/pop\_size$ and $2/pop\_size$.

The cumulative probability (Q) of each rank, for $rank = 1, ..., pop\_size$, is determined using $Q(rank) = \sum_{j=1}^{rank} Pr(j)$. The selection rule can then be implemented using the following steps.

Procedure: Selection rule

1. Sort the individuals in increasing order of their objective function values, and associate each individual with its corresponding $Q(rank)$ value.

2. Generate a set of random numbers $R'(i)$, for $i = 1, ..., pop\_size$, where each random number $R'(i)$ follows a uniform distribution between 0 and 1.

3. For $i = 1, ..., pop\_size$, do the following steps:

   (a) $x = \arg\min_{j} \{Q(j) : Q(j) > R'(i)\}$

   (b) $Pop'(i) = Pop(x)$

4.3.3 Genetic operators

After a new population is generated, we apply genetic operators on selected number of solutions. The basic types of genetic operators include the *mutation* and the *crossover*.

Mutation operator

Mutation operator alters one or more elements of a solution by some random amount to form an offspring. This injects genetic diversity into the population. Thus, it allows the algorithm to explore new regions of the search space. Several different mutation operators are used in the open literature [21]. They are referred

to as *inversion* operator, *insertion* operator, *displacement* operator, *reciprocal exchange* operator, and *heuristic mutation* operator. In small problem instances, we performed an experiment using the inversion operator, the insertion operator, and the reciprocal exchange operator. The results indicated that the insertion operator is more effective than other mutation operators in this context. Thus, in the following computational study, we employ the insertion operator.

To implement this operator, we define P_m as the mutation probability. A random number between 0 and 1 is then generated for each individual. If this number is less than P_m, we perform the insertion operator to this individual. The description of the insertion operator is shown as follow.

Procedure: Insertion operator

1. Select randomly an element of each selected individual i (say the element at the x^{th} position in individual i).

2. Insert this element to a randomly selected new position in i (say at the y^{th} position in i with $y \neq x$), and shift the position of all jobs between x and y by one unit accordingly.

Crossover operator

The purpose of crossover operator is to combine the good portions of the selected parents to create better children. However, the resulting children are not necessarily better than the parents. Such individuals will eventually diminish in the

subsequent generations. But, the good portions of these children can still be passed to later generations.

In implementing the crossover operator, we denote by P_c the probability that an individual is selected for this operation. For each individual, a random number between 0 and 1 is generated and compared with P_c. If that number is less than P_c, we select this individual as the candidate for the crossover operation. Then, we select K individuals of the selected candidates, where K is the largest even number which is less than or equal to the number of selected candidates. Of these K individuals, every two sequential individuals are paired up and the crossover operator is applied.

The crossover procedures that we consider here is the OX procedure as defined in [21]. We apply the OX procedure to the two selected individuals i and j, and generate two new individuals i' and j'. The description of the OX procedure is given below and the illustration of its implementation is shown in Figure 4.2.

Procedure: OX

1. Select randomly a substring from individual i.
2. Move this substring to i' at the same location.
3. Remove the jobs which are already in i' from individual j.
4. Move the remaining jobs in j from left to right to the empty position in i' starting from the first empty position in i'.

5. Repeat the same steps (step 1 to 4) starting with individual j to produce j'.

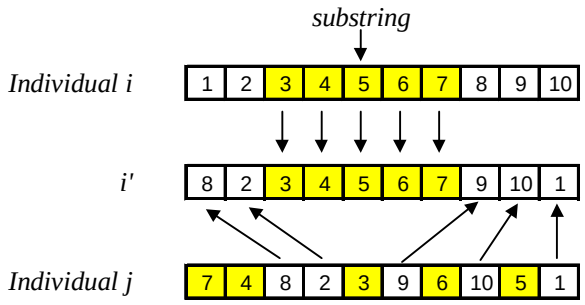


Figure 4.2: Illustration of the OX procedure to generate i'.

4.3.4 Parameter values

To obtain the parameter values for the genetic algorithm, we perform a preliminary study with a small number of problem instances. Based on this preliminary study, we obtain the following parameter values.

pop_size	=	100	(population size)
q	=	$\frac{1.7}{pop\_size}$	(selection mechanism control parameter)
P_c	=	0.5	(crossover probability)
P_m	=	0.3	(mutation probability)
$Count$	=	100	(the number of generations without improvement)
$Iter\_max$	=	1000	(the maximum number of generations)

4.4 Tabu Search algorithm

Tabu search algorithm has been widely applied for solving combinatorial optimization problems. The algorithm begins with an initial solution which can be generated either randomly or using the result of a known constructive procedure. The tabu search then improves the solution through a series of iteration. At each iteration, the tabu search investigates the neighborhood of the current solution. A move is defined by transition from the current solution to a selected neighbor. The process is then repeated until the stopping criterion is met.

The main feature of this algorithm that distinguishes it from a local improvement procedure is that it allows a move to a solution which is worse than the current solution. Several other features related with the tabu search include:

- It maintains a list of solution attributes (or move attributes) that must be avoided at any given iteration. This list is referred to as the tabu list.
- The tabu list is updated based on some memory structure (short term memory).
- The algorithm allows for exception from the tabu list if such a move leads to a promising solution (aspiration criterion).
- It remembers the features of the moves performed or the solutions visited so as to diversify or intensify the search process (long term memory).

There are several fundamental issues related to the implementation of tabu search algorithm. These include the neighborhood structure, tabu list, aspiration criterion, long term memory, and stopping criterion. These issues are discussed in the following subsections.

4.4.1 Neighborhood structure

In a preliminary investigation, we noted that neither of the three neighborhood structures F, L, or G that we discussed earlier (see page 36) is suitable in the context of the tabu search procedure. Neighborhood structures F and L are too limiting and did not produce good solutions, while neighborhood structure G resulted in excessive computational time due to its relatively larger size. However, the quality of the solutions obtained via the neighborhood structure G were relatively good.

In order to reduce the CPU time requirement of the search procedure at each iteration, while maintaining the fundamental features of the neighborhood structure G, we adopted two different search strategies. These two strategies are aimed at reducing the number of neighboring solutions considered at each iteration. Following is the description of each search strategy.

Search strategy 1

For each selected job for moving (i.e., the job at the x^{th} position in S, for $x = 1, ..., N$), we only consider one insertion position (i.e., the y^{th} position in S with $y \neq x$). We choose this position y randomly among the $N - 1$ possible insertion positions. This results in considering only N neighborhood points at each iteration. We notice that this strategy performs well for small size instances (as measured by the number of jobs). However, as the size of instances increases this strategy becomes inefficient. Therefore, for large size instances we employ the following strategy.

Search strategy 2

This strategy is similar with the first strategy above. But, at each iteration, instead of considering all N jobs for moving, we consider only a fraction of the N jobs. We divide the N positions in S into h equal subsets of consecutive positions of size θ (naturally, θ and h are chosen such that $\theta h = N$), and we denote each subset by τ_i, for $i = 1, ..., h$. At each iteration, we only consider moving the jobs which are in the subset that we consider at that iteration, starting from subset τ_1. For each of the jobs in this subset, we randomly choose an insertion position among the $N-1$ possible insertion positions. For the next iteration, we consider positions in subset τ_2, etc. Once we consider all subsets, we return to subset τ_1 and continue in the same manner. It follows that at each iteration, we consider only θ neighborhood

points. This type of strategy in the context of tabu search is described in [52].

In addition to the above strategies, we specify that the algorithm moves to the first non tabu neighbor that improves the current solution. If there is no such a move that improves the current solution, then the whole neighborhood is examined and the best non tabu neighbor is chosen for the next move. This method is employed by Widmer et al. [65] for the flowshop problem.

4.4.2 Tabu list and aspiration criterion

In order to avoid cycling, a tabu list is introduced during the search. A move to a neighbor is allowed only if the selected move is not in the tabu list. Throughout the iterations, the tabu list maintains the information of the moves that have been performed recently for a period of time tt, i.e., tabu tenure.

In our implementation, the tabu list stores the following elements: the job currently moved and its position before moving. Recall that a move to the selected neighbor is performed by removing a job j in the x^{th} position in S, and inserting it at the y^{th} position in S. Upon performing this move, we remove the last element of the tabu list and store j and x in the first position of the tabu list. In this manner, we do not allow job j to return to the x^{th} position in S for the next several iterations as defined by the tabu tenure, except when the aspiration criterion is met.

The aspiration criterion is the criterion which is used to override the tabu status of a move. By this criterion, we allow to choose a tabu move if such a move leads

to a neighboring solution which is better than the objective function value of the best solution obtained so far.

4.4.3 Exploration and exploitation of the solution space

In a tabu search algorithm, most of the computational effort is related to the neighborhood evaluation. It is our concern to balance the *exploration* and *exploitation* of the solution space. By exploration we mean the ability of the procedure to diversify the search throughout the solution space, while by exploitation we mean the ability of the procedure to intensify the search in the promising areas of the solution space (around the best solution found). Below, we discuss the implementation of the exploration and the exploitation strategies in our tabu search.

Exploration

In order to explore the solution space, throughout the search we maintain a matrix E of size N by N. The entry E_{jx}, for $j = 1, ..., N$ and $x = 1, ..., N$, maintains the frequency of moving job j from the x^{th} position in S. We use this frequency as a penalty measure for the neighborhood points evaluated at the current iteration. As discussed in section 4.1 of [22] and in section 3.2 of [54], this form of penalty function in the context of tabu search is commonly used. Its implementation can be described as follow.

At the start of the search, we initialize all entries of this matrix to zero. Upon

moving job j in the x^{th} position in S and inserting it at the y^{th} position in S, i.e., move $v = (x, y)$, we update E_{jx} by setting $E_{jx} = E_{jx} + 1$. The value of E_{jx} is used as the penalty for the move $v = (x, y)$. In subsequent iterations, upon performing move $v = (x, y)$, we define a dummy objective function value C'_{max} as follow: $C'_{max} = C_{max} + E_{jx}$, where C_{max} is the objective function value of move $v = (x, y)$. Then, the selection of a neighbor for the next move is based on C'_{max}. In this manner, we expect that we increase the chance to move to a neighborhood that has not been previously explored.

Exploitation

As mentioned in subsection 4.4.1, for each job that we consider for moving, we randomly select one position y for its insertion. If a move yields a better solution than the best solution so far (as measured by C_{max}), we intensify the search around this solution. This exploitation is performed by generating randomly k insertion positions for each job considered (instead of only one). If a move does not yield a better solution than the best solution so far, we return $k = 1$.

4.4.4 Starting solution and stopping criterion

The implementation of the tabu search algorithm to our problem is started by defining an initial solution and evaluating the makespan of the initial solution. In this study, we generate the initial solution randomly. The search is then continued

by evaluating the neighborhood of the current solution using the search strategies that we discussed above. We repeat the iteration until the stopping criterion is met. As for the stopping criterion, we terminate the algorithm if there is no improvement to the best solution obtained after a certain number of iterations ($Count$). We also limit the maximum number of iterations ($iter\_max$).

4.4.5 Parameter values

To obtain the parameter values for the tabu search algorithm, we perform a preliminary study with a small number of problem instances. Based on this preliminary study, we obtain the following parameter values.

k	=	5	(exploitation factor)
tt	=	8	(tabu tenure)
$Count$	=	100	(number of iterations without improvement)
$Iter\_max$	=	1000	(maximum number of iterations)
θ	=	$Min(N, 50)$[1]	(the length of job fraction in each iteration)

[1]If $N > 50$ and N is not divisible by 50, we set the length of the last subset equals to $N \pmod{50}$.

Chapter 5

Computational Experiments and Results for FSPM Problem

In this chapter, we present the results of a computational experiment that we performed with the procedures discussed earlier. All of these algorithms are coded using Fortran90 and ran on a 266 Mhz PC.

In section 5.1, we describe the data sets that we used in the experiment. In section 5.2, we present the method that we developed to determine the lower bound to the optimal makespan. In section 5.3, we present the experimental results to show the impact of different strategies within each procedure. Finally, in section 5.4, we present an empirical study of the quality of the solutions.

5.1 Data sets

Since we do not have access to the real world (benchmark) data, we follow the common practice and randomly generate a set of instances of the problem. The generation of each instance includes the processing time of each job at each stage. For the purpose of our study, we define two different groups of instances designated as data sets of type I and type II. An instance of type I data set has a special structure that allows us to determine its corresponding optimal makespan, for comparative purposes. An instance of type II data set does not have this property, and we do not know its corresponding makespan.

In the following discussion, we describe how we construct the problem instances of each type.

5.1.1 Data set of type I

For this set of instances, we generate the processing times ($p_{j\ell}$) randomly, but the range of values for each processing time in each instance is chosen in such a manner that the corresponding optimal makespan can be easily determined. The details of the procedure for generating the job processing times for an instance of the problem in this group, along with the procedure for determining its optimal makespan, are shown in appendix A.

For this type of data set, we generate instances of different sizes with the number of jobs $N = 20, 50, 100, 150$ and the number of stages $L = 2, 4, 6, 8$. We assume

that all stages have identical number of machines with $m_\ell = 2, 4, 6$, for $\ell = 1, ..., L$. For each problem size, we construct and solve 5 instances of the problem using each of the pertinent procedures, and present the average makespan obtained for the 5 instances.

5.1.2 Data set of type II

For these set of instances, we randomly generate the job processing time as an integer from the uniform distribution over the interval $[1, 100]$. This method is used by Nowicki et al. [44]. For this type of data set, we generate instances of different sizes similar to the sizes that we stated for data set of type I, i.e., with $N = 20, 50, 100, 150$, $L = 2, 4, 6, 8$, and $m_\ell = 2, 4, 6$, for $\ell = 1, ..., L$. For each problem size, we also construct and solve 5 instances and present the average makespan of the 5 instances. The rationale for generating the job processing time in this manner and choosing the size of the instances are discussed later in page 96.

In addition, we also perform an experiment using the most suitable algorithm (which is considered the best for the FSPM problem among the proposed algorithms) with instances of this type with different machine configuration, i.e., different number of machines at different stages. In this type of experiment, we generate instances of different sizes with the number of jobs (N) and stages (L) similar to above. The number of machines at each stage is generated randomly following a

uniform distribution between 1 and 5, i.e., $P(m_\ell = k) = \frac{1}{5}$, for $k = 1, ..., 5$, and for all ℓ. For each problem size, we construct and solve 10 instances and present the average makespan of the 10 instances. We have adopted this strategy so that our results would be comparable with those discussed in [44].

Thus, for the experiment, we generate 240 different instances of data set of type I and 400 different instances of data set of type II.

5.2 Lower bounds

In order to measure the solution quality of the instances with unknown optimal makespan, we compare the makespan obtained for each instance of the problem with its corresponding lower bound for the optimal value. In the following discussion, we present three different lower bounds that we developed for the L-stage FSPM problem. The first two lower bounds are based on relaxing the restrictions imposed by the number of machines at different stages. The third lower bound is obtained by removing these relaxations. The first and the second lower bounds were also independently developed in [64].

5.2.1 The first lower bound

The first lower bound is obtained by relaxing the problem, i.e., by assuming that the number of machines at each stage is at least equal to the number of jobs. Thus, at each stage, each job can be assigned to a separate machine. As soon as this job

is completed at a stage, it can immediately be processed at the next stage. The completion time of the last job at the last stage is the optimal makespan to the relaxed problem and it is a lower bound for the optimal makespan of the original problem. We refer to this lower bound as LB_1,

$$LB_1 = \max_{j \in \{1,...,N\}} \left\{ \sum_{\ell=1}^{L} p_{j\ell} \right\}.$$

5.2.2 The second lower bound

To determine the second lower bound, we consider a stage ℓ, for $\ell = 1, ..., L$, as the basis of finding a lower bound. Given a stage ℓ, we relax the problem by assuming that the number of machines at stages other than stage ℓ is greater than or equal to the number of machines at stage ℓ, i.e., $m_k \geq m_\ell$ for all $k \neq \ell$. Let

$$P_{j,a\to b} = p_{j,a} + p_{j,a+1} + ... + p_{j,b}$$

$$P'_{x,a\to b} = \text{the } x^{th} \text{ shortest of } P_{j,a\to b} \quad \text{(all ties are broken arbitrarily).}$$

Following the flowshop constraint, it must be true that each machine at stage ℓ (for $\ell > 1$) must be idle for some period of time before it starts processing a job. This is due to the flowshop requirement that before a job can be started at stage ℓ, it must have been processed at all stages $k < \ell$. Therefore, it is obvious that at stage ℓ, there will be a machine with idle time no less than $P'_{1,1\to(\ell-1)}$, a machine with idle time no less than $P'_{2,1\to(\ell-1)}$, ..., and a machine with idle time no

less than $P'_{m_\ell,1\rightarrow(\ell-1)}$. The minimum total idle time of the machines at stage ℓ (for $\ell > 1$) before they start processing a job is the summation of such idle times, i.e., $\sum_{x=1}^{m_\ell} P'_{x,1\rightarrow(\ell-1)}$.

Defining α_ℓ as the average idle time of the machines at stage ℓ before they start processing a job, we obtain α_ℓ as follow.

$$\alpha_\ell = \begin{cases} \frac{1}{m_\ell} \sum_{x=1}^{m_\ell} P'_{x,1\rightarrow(\ell-1)} & \text{, if } \ell > 1 \\ 0 & \text{, if } \ell = 1 \end{cases}$$

By symmetry, we determine that each machine at stage ℓ (for $\ell < L$) must be idle for some period of time after completing the last job up to the end of operation at stage L. Defining β_ℓ as the average idle time of the machines at stage ℓ after they complete the last job, we obtain β_ℓ using the following formula.

$$\beta_\ell = \begin{cases} \frac{1}{m_\ell} \sum_{x=1}^{m_\ell} P'_{x,(\ell+1)\rightarrow L} & \text{, if } \ell < L \\ 0 & \text{, if } \ell = L \end{cases}$$

Letting T_ℓ be the average workload of each machine at stage ℓ, we obtain T_ℓ as follow.

$$T_\ell = \frac{1}{m_\ell} \sum_{j=1}^{N} p_{j\ell}$$

Observe that T_ℓ, α_ℓ, and β_ℓ are the minimum values in their corresponding contexts. Therefore, it is obvious that the summation of T_ℓ, α_ℓ, and β_ℓ must be

less than or equal to the optimum makespan. Thus, the second lower bound LB_2 can be obtained as follow.

$$LB_2 = \max_{\ell \in \{1,\ldots,L\}} \{\alpha_\ell + T_\ell + \beta_\ell\}$$

As we mentioned earlier, both LB_1 and LB_2 were also independently developed by Vandevelde [64]. She refers to LB_1 as *the job based bound* and LB_2 as *the set based bound*. The exact form of LB_2 that Vandevelde proposes is slightly different from what we stated above. She proposes

$$LB_2' = \max_{\ell \in \{1,\ldots,L\}} \lceil \alpha_\ell + T_\ell + \beta_\ell \rceil.$$

Although LB_2' is a valid lower bound if all processing times are integer (which is the assumption used by Vandevelde in developing LB_2'), it is not necessarily valid if some of the processing times are not integer. The following example demonstrate that LB_2' is not valid if some of the processing times are not integer.

Example 1

Consider an instance of 3-stage FSPM problem with 4 jobs, and the number of machines $m_1 = m_2 = m_3 = 2$. The processing time of each job at each stage is given as follow: $p_{j1} = 2, p_{j2} = 3$, and $p_{j3} = 3\frac{1}{4}$, for all j. As shown in Figure 5.1, it is obvious that the optimal makespan is $11\frac{1}{2}$. Using LB_2, we determine that $LB_2 = 11\frac{1}{2}$, which is equal to the optimal makespan. On the other hand, we obtain $LB_2' = \lceil 11\frac{1}{2} \rceil = 12$, which is greater than the optimal makespan.□

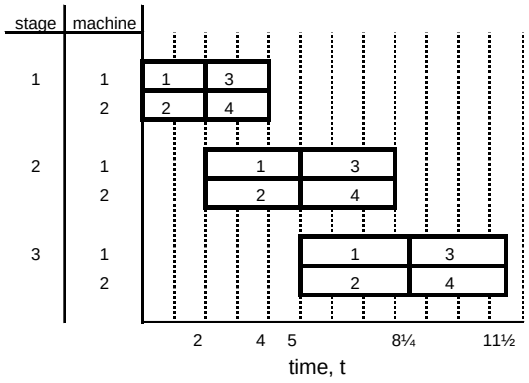


Figure 5.1: An example of schedule where LB_2' of [64] is greater than the optimal makespan.

It is clear that LB_2 can be improved by removing some of the relaxations that we made, i.e., the case when the number of the machines at different stages are not the same. In the following subsection, we present the third lower bound that we developed, which is an improvement to LB_2.

5.2.3 The third lower bound

As we mentioned above, in developing LB_2, we assume that the number of machines at stage k, for $k \neq \ell$, is greater than or equal to the number of machines at stage ℓ, i.e., $m_k \geq m_\ell$ for $k \neq \ell$. As such, to determine α_ℓ (as in the formulation for LB_2) we have assumed that each of the jobs which are processed first on the m_ℓ machines at stage ℓ, can be assigned to a separate machine at each stage k, for $k < \ell$. As soon as this job is completed at a stage, it can be immediately processed at the

next stage. As such, $\sum_{x=1}^{m_\ell} P'_{x,1\rightarrow(\ell-1)}$ is the minimum total idle time of the machines at stage ℓ (for $\ell > 1$) before they start processing a job. A similar assumption with regard to the number of machines after stage ℓ leads to the value of β_ℓ as described above.

Now, we consider the case when there exists a stage k, for $k < \ell$, such that $m_k < m_\ell$ (i.e., we remove the relaxation on the number of machines that we assumed in determining LB_2). In this case, we must consider the fact that in addition to α_ℓ, there must be an additional idle time to the machines at stage ℓ before they start processing the first job. This is due to the fact that some of the jobs which are processed first on the m_ℓ machines at stage ℓ can not be assigned to a separate machine at stage k, for $k < \ell$. To describe such situation, consider the argument below. In the subsequent discussion, we define $p'_{x,a}$ as follow.

$$p'_{x,a} = \text{the } x^{th} \text{ shortest processing time at stage } a \text{ (all ties are broken arbitrarily).}$$

Let s be the largest index of a stage such that $s < \ell$ and $m_s < m_\ell$, and let $d_{\ell s} = m_\ell - m_s$. Since $m_s < m_\ell$, it must be true that $d_{\ell s}$ jobs must wait until the first m_s jobs have been completed at stage s before they can be started at stage s. Therefore, it is obvious that at stage s, there must be a job which must wait for at least $p'_{1,s}$ unit times, a job which must wait for $p'_{2,s}$ unit times, ..., and a job which must wait for $p'_{d_{\ell s},s}$ unit times, before they can be started at stage s. The minimum total waiting time of these $d_{\ell s}$ jobs is the summation of such waiting

times, i.e., $\sum_{x=1}^{d_{\ell s}} p'_{x,s}$. Following the flowshop rule, the delay due to these $d_{\ell s}$ jobs that must wait before they can be started at stage s will also cause a delay to the machines at stage ℓ. Therefore, the minimum total idle time at stage ℓ will not be shorter than

$$\sum_{x=1}^{m_\ell} P'_{x,1\to(\ell-1)} + \left(\sum_{x=1}^{d_{\ell s}} p'_{x,s}\right).$$

Now, we observe the jobs which are processed first at stage s. If the number of machines at stage k, for $k < s$, is greater than or equals to the number of machines at stage s, i.e., $m_k \geq m_s$ for $k < s$, then we will have the similar condition as the condition that we assumed in obtaining LB_2. At each stage $k < s$, the jobs which are processed first at stage s can be assigned to separate machines at stage k. Therefore, upon completion of their processing at each stage, these job can be started immediately at the next stage. However, if there exists a stage k, for $k < s$, such that $m_k < m_s$ (and necessarily $m_k < m_s < m_\ell$), the situation will be similar to the situation as we described for stage s with respect to stage ℓ above. Some of the jobs which are processed first on the m_s machines at stage s can not be assigned to separate machines at stage k, for $k < s$. Below, we describe such situation.

Let r be the largest index of a stage such that $r < s$ and $m_r < m_s$, and let $d_{sr} = m_s - m_r$. If there exist such stage r, using a similar argument as above, it must be true that there are d_{sr} jobs which must wait before they can be started

at stage r. The minimum total waiting time of these d_{sr} jobs before they can be started at stage r is $\sum_{x=1}^{d_{sr}} p'_{x,r}$. Since these d_{sr} jobs are delayed by $\sum_{x=1}^{d_{sr}} p'_{x,r}$ unit times at stage r, the starting time of these d_{sr} jobs at stage s will also be delayed with a total amount of $\sum_{x=1}^{d_{sr}} p'_{x,r}$ unit times. Upon completion at stage s, and since $m_s < m_\ell$, these d_{sr} jobs can be started immediately at the subsequent stages until stage ℓ. It follows that at stage ℓ, the starting time of these d_{sr} jobs are delayed with a total amount of $\sum_{x=1}^{d_{sr}} p'_{x,r}$ unit times. Taking into account the fact that $m_r < m_s < m_\ell$, it is clear that the minimum total idle time at stage ℓ will not be shorter than

$$\sum_{x=1}^{m_\ell} P'_{x,1\to(\ell-1)} + \left(\sum_{x=1}^{d_{\ell s}} p'_{x,s} + \sum_{x=1}^{d_{sr}} p'_{x,r}\right).$$

In a similar manner we repeat the process until there is no more stage $k < \ell$ with the number of machines less than the number of machines at the stage currently considered, and the results follow.

From the above argument, let us define γ_ℓ as the *minimum total delay* of the machines at stage ℓ due to the fact that some jobs which are processed first on the machines at stage ℓ can not be assigned to a separate machines at stage k, for $k < \ell$ and $m_k < m_\ell$. We obtain γ_ℓ using the following procedure.

Procedure: obtaining γ_ℓ

Step 1 - Set $\gamma_\ell = 0$ and $k = \ell$.

Step 2 - Let s be the largest stage such that $s < k$ and $m_s < m_k$; let $d_{ks} = m_k - m_s$. If there is no such stage s, terminate the procedure. Otherwise, calculate γ_ℓ as follow.

$$\gamma_\ell \longleftarrow \gamma_\ell + \sum_{x=1}^{d_{ks}} p'_{x,s}$$

Step 3 - Set $k = s$ and go to step 2.

Defining α'_ℓ as the average idle time of the machines at stage ℓ before they start processing a job, we obtain α'_ℓ as follow.

$$\alpha'_\ell = \begin{cases} \frac{1}{m_\ell}\left(\sum_{x=1}^{m_\ell} P'_{x,1\rightarrow(\ell-1)} + \gamma_\ell\right) & \text{, if } \ell > 1 \\ 0 & \text{, if } \ell = 1 \end{cases}$$

By symmetry, we determine the additional idle time due to the stages k, for $k > \ell$, such that $m_k < m_\ell$. Let us define ϕ_ℓ as the *minimum total idle time* of the machines at stage ℓ due to the stages k, for $k > \ell$ and $m_k < m_\ell$. We obtain ϕ_ℓ using the following procedure.

Procedure: obtaining ϕ_ℓ

Step 1 - Set $\phi_\ell = 0$ and $k = \ell$.

Step 2 - Let s be the smallest stage such that $s > k$ and $m_s < m_k$; let $d_{ks} = m_k - m_s$. If there is no such stage s, terminate the procedure. Otherwise, calculate ϕ_ℓ as follow.

$$\phi_\ell \longleftarrow \phi_\ell + \sum_{x=1}^{d_{ks}} p'_{x,s}$$

Step 3 - Set $k = s$ and go to step 2.

Defining β'_ℓ as the average idle time of the machines at stage ℓ after they complete the last work, we obtain β'_ℓ using the following formula.

$$\beta'_\ell = \begin{cases} \dfrac{1}{m_\ell}\left(\sum_{x=1}^{m_\ell} P'_{x,(\ell+1)\to L} + \phi_\ell\right) & \text{, if } \ell < L \\ 0 & \text{, if } \ell = L \end{cases}$$

From the above argument, we can observe that α'_ℓ, and β'_ℓ are the minimum values in their corresponding contexts. Therefore, it is obvious that the summation of T_ℓ, α'_ℓ, and β'_ℓ must be less than or equal to the optimum makespan (where T_ℓ is defined similarly as in the context of LB_2). Thus, the third lower bound LB_3 can be obtained as stated in the following proposition.

Proposition 1 $LB_3 = \max_{\ell \in \{1,\ldots,L\}} \{\alpha'_\ell + T_\ell + \beta'_\ell\}$ *is a lower bound for the L-stage FSPM problem.*

We observe that if the number of machines at all stages are the same, then LB_3 is identical to LB_2. However, if the number of machines at different stages are not the same, then LB_2 is dominated by LB_3, i.e., $LB_2 < LB_3$, since LB_2 does not include the γ_ℓ and ϕ_ℓ terms. Since in all situations we have $LB_2 \leq LB_3$, we can state the best lower bound LB as

$LB = \max\{LB_1, LB_3\}$.

5.2.4 A comparison with other lower bounds

In this subsection, we compare the lower bounds that we have developed, particularly LB_3, with other lower bounds from the open literature. For the L-stage FSPM problem, we have shown earlier that LB_2 which is similar to that developed by Vandevelde [64] is dominated by LB_3. (We should notice that in addition to the job based bound and the set based bound, Vandevelde also developed several other lower bounds which can not be directly compared with LB_3).

For the 2-stage FSPM problem, Lee and Vairaktarakis [32] have developed a set of lower bounds as follow.

$$LV_1 = \frac{1}{m_2}\left\{\sum_{x=1}^{m_2} p'_{x,1} + \sum_{x=1}^{N} p'_{x,2}\right\} \quad \text{, for } m_1 \geq m_2$$

$$LV_2 = \frac{1}{m_2}\left\{\sum_{x=1}^{m_2} p'_{x,1} + (m_2 - m_1)p'_{1,1} + \sum_{x=1}^{N} p'_{x,2}\right\} \quad \text{, for } m_1 < m_2$$

$$LV_3 = \frac{1}{m_1}\left\{\sum_{x=1}^{m_1} p'_{x,2} + (m_1 - m_2)p'_{1,2} + \sum_{x=1}^{N} p'_{x,1}\right\} \quad \text{, for } m_1 > m_2$$

$$LV_4 = \frac{1}{m_1}\left\{\sum_{x=1}^{m_1} p'_{x,2} + \sum_{x=1}^{N} p'_{x,1}\right\} \quad \text{, for } m_1 \leq m_2$$

For $L = 2$, we observe that for $m_1 \geq m_2$, LB_3 that we have developed reduces to the similar formulation as LV_1, and for $m_1 \leq m_2$, LB_3 reduces to the similar formulation as LV_4. To compare with LV_2 and LV_3, we note that for $m_1 < m_2$ and $m_1 > m_2$, the respective components of LB_3 reduce to the following formulation,

respectively:

$$LB_3(\ell = 2) = \frac{1}{m_2}\left\{\sum_{x=1}^{m_2} p'_{x,1} + \sum_{x=1}^{m_2-m_1} p'_{x,1} + \sum_{x=1}^{N} p'_{x,2}\right\} \quad \text{, for } m_1 < m_2$$

$$LB_3(\ell = 1) = \frac{1}{m_1}\left\{\sum_{x=1}^{m_1} p'_{x,2} + \sum_{x=1}^{m_1-m_2} p'_{x,2} + \sum_{x=1}^{N} p'_{x,1}\right\} \quad \text{, for } m_1 > m_2.$$

Comparing the formulation for $LB_3(\ell = 2)$ with LV_2 and the formulation for $LB_3(\ell = 1)$ with LV_3, we can observe the following.

$$\sum_{x=1}^{m_2-m_1} p'_{x,1} \geq (m_2 - m_1)p'_{1,1} \qquad \text{, for } m_1 < m_2$$

and

$$\sum_{x=1}^{m_1-m_2} p'_{x,2} \geq (m_1 - m_2)p'_{1,2} \qquad \text{, for } m_1 > m_2$$

Thus, from the above observations, we can see that for $L = 2$, LB_3 is stronger than the lower bounds of Lee and Vairaktarakis, although it requires more calculations. (Again, we should notice that in addition to the above lower bounds LV_1 through LV_4, Lee and Vairaktarakis also consider the makespan of an auxiliary problem as a lower bound, where the auxiliary problem is defined as a 2-stage FSPM problem with one machine at each stage. The processing time of job j at stage 1 is defined as $\frac{p_{j,1}}{m_1}$, and at stage 2 is defined as $\frac{p_{j,2}}{m_2}$, for $j = 1$ to N. The makespan of the auxiliary problem C_{LB} can then be obtained by solving the auxiliary problem using Johnson's algorithm [27]. As such, LB_3 can not be directly compared with this lower bound, and in some instances with $L = 2$, it can be dominated).

5.3 A relative comparison of the proposed procedures

In the context of the proposed algorithms with vector representation of the solution (i.e., local improvement, genetic algorithm, and tabu search) we have two different procedures, namely procedures $Z1$ and $Z2$, to construct a complete schedule. In addition, we also defined three different types of neighborhood structures (i.e., G, F, and L) in the context of local improvement procedure using vector representation. With regard to these options, in this part of our experiment we have two specific objectives.

1. Find the most effective set of options in the context of each proposed search procedure with vector representation.

2. Compare the results obtained via each of these procedures (at its most effective set of options) with the results obtained via the search method using matrix representation. Our objective here is to determine the most effective algorithm among the algorithms that we proposed for solving the FSPM problem.

In this section, we use the makespans obtained by each procedure in each instance as the basis for comparison. In performing the experiment in this section, we divided the experiment into two parts which are distinguished by the manner we obtain the makespan.

In the first part of the experiment, the makespans are obtained upon running the program for a specific amount of CPU time. Here, we omit the stopping criterion defined for the genetic algorithm and tabu search procedure (as defined in sections 4.3 and 4.4, respectively). The experimental results are shown in subsections 5.3.1 to 5.3.4.

In the second part of the experiment, we further investigate the algorithms. Here, we run the programs until the specified stopping criterion is met and retain the best result. The results of this experiment are presented in subsection 5.3.5.

5.3.1 Local improvement with vector representation

In chapter 4, we defined three different neighborhood structures (i.e., G, F, and L) and two different methods for constructing a complete schedule for a given permutation vector (i.e., procedures $Z1$ and $Z2$). Our objective here is to compare these variations on an empirical basis, and decide which option is more effective for the local improvement procedure.

Considering the options stated above, we have 6 different variations of the local improvement procedure (namely, LI-Z1-G, LI-Z2-G, LI-Z1-F, LI-Z2-F, LI-Z1-L, and LI-Z2-L, where LI refers to the local improvement procedure, Z1 and Z2 refer to the procedure used to construct the complete solution for a given vector, and G, F, and L refer to the neighborhood structure employed).

In this study, we solve the instances of each type of data set using each of

the above variations of the local improvement procedure. In order to make an objective comparison, we run each program with the same amount of CPU time (i.e., 30 seconds) using as many randomly generated starting points as possible within this limit, and retain the best result. We use 30 seconds as the time limit in this study, since it is short enough to allow completion of the study within a reasonable amount of time, and yet it is long enough to allow a reasonable number of starting points generated for the local improvement. We report the average number of starting points generated within this time limit in appendix B.

The results for data set of type I and II are presented in Tables 5.1 and 5.2, respectively. The first column in each table shows the size of the instance (the number of jobs, the number of stages, and the number of machines at each stage). In each of the remaining columns, we present the average makespan obtained for five instances of the problem using a variation of the algorithm.

From these tables, we make the following observations in the context of LI-Z1.

1. From Table 5.1, we can see that using data set of type I, algorithm LI-Z1-F gives a smaller average makespan than algorithms LI-Z1-G and LI-Z1-L. We also run the Wilcoxon signed rank test to compare the results obtained by each pair of algorithms. Letting μ_i to be the average makespan obtained by algorithm i, we use the null hypothesis Ho: $\mu_i = \mu_j$ with the alternative hypothesis $\mu_i \neq \mu_j$. We discover that at 99% confidence level, the null hypothesis cannot be rejected for the pair of algorithms LI-Z1-F and LI-

Z1-L. On the other hand, we reject the null hypothesis at 99% confidence level for the following pairs of algorithms: LI-Z1-L vs LI-Z1-G and LI-Z1-F vs LI-Z1-G. Thus, there is a strong statistical evidence that the makespans obtained via LI-Z1-G are larger than those obtained via LI-Z1-L or LI-Z1-F, but there is no such strong evidence in comparing the results of LI-Z1-F and LI-Z1-L. Comparing the makespan for each problem size, we observe that in 30 out of 48 instances, LI-Z1-L obtains smaller makespan than LI-Z1-F. We conclude that algorithm LI-Z1-L is more effective in this context, although the results obtained using LI-Z1-F are quite close.

2. From Table 5.2, using data set of type II, we observe that LI-Z1-L gives a smaller average makespan than LI-Z1-G and LI-Z1-F. In addition, in 47 out of 48 instances, LI-Z1-L yields smaller makespan than LI-Z2-F, and in 32 out of 48 instances LI-Z1-L obtains smaller makespan than LI-Z1-G. A statistical test analysis similar to the one discussed above (at 99% confidence level) also supports the observation that neighborhood structure L is more appropriate to use in this context.

From these observations, we conclude that LI-Z1-L is more effective than both LI-Z1-G and LI-Z1-F in problem instances that we considered.

In the context of LI-Z2, we make the following observations.

3. From Table 5.1, we can observe that using data set of type I, algorithm LI-Z2-L yields a smaller average makespan than either algorithm LI-Z2-F

or LI-Z2-G. In 30 out of 48 instances, LI-Z2-L gives smaller makespan than LI-Z2-F, and in 38 out of 48 instances, LI-Z2-L obtains smaller makespan than LI-Z2-G. The Wilcoxon signed rank test supports these findings. We conclude that in this context, neighborhood structures L is more effective than both neighborhood structure G and F.

4. Using data set of type II, as shown in Table 5.2, we can observe that LI-Z2-L yields a smaller average makespan than either LI-Z2-G or LI-Z2-F. Comparison on the makespan of each problem size shows that in 44 out of 48 instances, LI-Z2-L obtains smaller makespan than LI-Z2-F, and in 32 out of 48 instances, LI-Z2-L yields smaller makespan than LI-Z2-G. The Wilcoxon signed rank test also supports these findings. Therefore, we conclude that for this type of instances, neighborhood structure L is more appropriate.

From these observations, we conclude that LI-Z2-L performs better than both LI-Z2-G and LI-Z2-F in problem instances that we considered.

To determine which algorithm is more effective between algorithms LI-Z1-L and LI-Z2-L, we compare the makespans obtained using these two algorithms and we make the following observations.

- Using data set of type I, LI-Z2-L yields a smaller average makespan than LI-Z1-L. However, a statistical test analysis similar to the one discussed above shows that we are not able to reject the null hypothesis at 99% confidence

level. Besides, in 26 out of 48 instances, LI-Z1-L obtains smaller makespan than LI-Z2-L.

- Using data set of type II, LI-Z1-L gives smaller average makespan than LI-Z2-L. In addition, in 38 out of 48 instances, LI-Z1-L gives smaller makespan than LI-Z2-L. A similar statistical test analysis also supports this finding.

Thus, based on these two observations, we choose algorithm LI-Z1-L as the most effective implementation of the local improvement method.

5.3.2 Genetic algorithm with vector representation

In the context of the genetic algorithm, similar to the local improvement algorithm, we have two procedures $Z1$ and $Z2$ to construct a complete schedule for a given permutation vector. We refer to the resulting algorithms as GA-Z1 and GA-Z2, respectively, where GA refers to the genetic algorithm and Z1 and Z2 refer to the procedure used. In this part of the experiment, our objective is to determine which algorithm is more effective.

For the experiment, we use the same instances as we used earlier. For each problem instance, we run the programs for 30 seconds of CPU time (omitting the stopping criterion as defined earlier in section 4.3) and report the best makespan obtained. Again, we use a time limit of 30 seconds of CPU time here for practical reason. In appendix B, we report the number of population generations performed within this time limit. The results of the experiment for data sets of type I and II

Table 5.1: The makespans obtained using LI-Z1 and LI-Z2 with neighborhood structures G, F, and L (data set of type I).

N x L x m_L	LI-Z1-F	LI-Z1-G	LI-Z1-L	LI-Z2-F	LI-Z2-G	LI-Z2-L
20 x 2 x 2	1187.40	1181.60	1191.60	1186.20	1181.20	1190.20
20 x 2 x 4	507.20	504.60	506.80	507.40	505.20	507.60
20 x 2 x 6	352.20	351.20	350.20	352.60	351.20	351.60
20 x 4 x 2	1313.60	1306.60	1316.60	1309.20	1301.80	1325.00
20 x 4 x 4	559.60	555.40	557.80	560.40	556.80	558.40
20 x 4 x 6	430.60	428.60	429.20	430.20	429.00	428.40
20 x 6 x 2	1371.20	1353.40	1384.40	1371.40	1357.40	1384.80
20 x 6 x 4	611.80	608.80	603.80	611.40	606.40	606.00
20 x 6 x 6	534.60	531.20	527.80	531.60	530.40	529.00
20 x 8 x 2	1315.40	1305.80	1334.80	1322.00	1308.00	1334.80
20 x 8 x 4	658.80	653.80	652.40	654.00	651.80	656.60
20 x 8 x 6	498.40	496.60	493.40	500.60	500.20	497.00
50 x 2 x 2	4743.40	4743.20	4750.60	4739.20	4739.40	4751.40
50 x 2 x 4	1660.80	1659.00	1660.00	1661.00	1663.60	1658.80
50 x 2 x 6	948.00	947.00	948.80	949.80	948.00	949.00
50 x 4 x 2	5205.40	5297.20	5237.60	5188.20	5298.60	5240.60
50 x 4 x 4	1848.20	1866.40	1843.20	1858.60	1857.80	1843.00
50 x 4 x 6	1007.40	1014.20	996.40	1009.20	1010.20	999.80
50 x 6 x 2	5274.60	5483.20	5317.20	5281.20	5506.80	5319.20
50 x 6 x 4	1798.20	1841.40	1783.00	1814.40	1835.60	1787.00
50 x 6 x 6	1172.40	1202.20	1149.20	1168.80	1172.40	1146.60
50 x 8 x 2	5124.80	5455.00	5194.80	5128.80	5450.60	5168.40
50 x 8 x 4	1918.40	1982.80	1909.20	1926.80	2007.40	1908.20
50 x 8 x 6	1117.40	1108.20	1089.40	1110.80	1130.60	1086.40
100 x 2 x 2	15790.00	15893.20	15850.00	15790.00	15887.60	15852.20
100 x 2 x 4	4917.20	4932.60	4911.00	4910.40	4931.20	4917.00
100 x 2 x 6	2549.60	2553.60	2545.20	2551.60	2553.40	2547.00
100 x 4 x 2	16367.80	16802.20	16537.60	16385.60	16802.40	16546.20
100 x 4 x 4	5338.20	5457.20	5340.80	5366.60	5443.20	5342.40
100 x 4 x 6	2860.80	2923.80	2841.00	2877.60	2925.60	2842.80
100 x 6 x 2	17152.00	18021.80	17381.00	17149.60	18013.60	17367.20
100 x 6 x 4	5518.20	5671.00	5432.60	5500.60	5665.60	5444.60
100 x 6 x 6	3019.00	3082.40	2947.20	3017.00	3058.00	2934.20
100 x 8 x 2	17057.20	18061.20	17181.00	16849.60	18054.40	17141.80
100 x 8 x 4	5781.80	6038.20	5700.20	5713.80	6045.80	5633.00
100 x 8 x 6	3147.80	3304.20	3043.20	3130.20	3279.00	3021.40
150 x 2 x 2	35086.80	35293.40	35207.60	35071.00	35293.40	35205.20
150 x 2 x 4	9652.20	9683.00	9655.40	9653.80	9681.40	9660.00
150 x 2 x 6	4936.60	4951.40	4934.60	4936.00	4950.60	4931.00
150 x 4 x 2	35163.80	35893.80	35427.00	35263.20	35907.60	35417.20
150 x 4 x 4	10208.60	10361.60	10137.20	10192.40	10363.60	10164.20
150 x 4 x 6	5211.60	5248.40	5173.00	5219.20	5264.40	5177.40
150 x 6 x 2	36137.00	36861.80	36322.60	36191.80	36874.00	36191.00
150 x 6 x 4	10930.60	11077.60	10806.60	10943.00	11083.60	10776.60
150 x 6 x 6	5442.00	5560.80	5379.00	5455.00	5551.80	5351.40
150 x 8 x 2	36742.80	37495.20	36596.60	36576.60	37457.40	36568.00
150 x 8 x 4	11290.00	11665.20	11112.20	11241.40	11621.80	11010.80
150 x 8 x 6	5431.60	5554.40	5294.80	5454.60	5560.80	5297.00
Average	7226.94	7380.53	7228.87	7221.13	7378.55	7220.15

Table 5.2: The makespans obtained using LI-Z1 and LI-Z2 with neighborhood structures G, F, and L (data set of type II).

N x L x m_L	LI-Z1-F	LI-Z1-G	LI-Z1-L	LI-Z2-F	LI-Z2-G	LI-Z2-L
20 x 2 x 2	550.00	550.00	550.00	550.00	550.00	550.00
20 x 2 x 4	290.40	282.60	287.80	288.60	283.00	287.80
20 x 2 x 6	227.80	222.40	225.00	228.20	222.80	225.80
20 x 4 x 2	642.60	639.60	640.60	654.20	640.60	647.80
20 x 4 x 4	412.00	403.80	407.00	414.40	402.60	409.00
20 x 4 x 6	323.20	316.00	318.80	324.40	316.80	317.80
20 x 6 x 2	792.80	777.60	785.40	803.40	780.40	788.40
20 x 6 x 4	505.80	496.00	502.20	509.40	498.00	505.00
20 x 6 x 6	439.80	439.20	439.20	442.60	439.80	440.40
20 x 8 x 2	943.80	923.40	934.80	948.00	932.20	938.40
20 x 8 x 4	609.60	600.20	601.20	615.60	601.20	606.40
20 x 8 x 6	556.20	550.40	550.00	559.40	549.80	553.00
50 x 2 x 2	1326.60	1325.00	1325.20	1327.80	1325.00	1325.20
50 x 2 x 4	668.60	660.40	661.60	668.40	660.40	662.20
50 x 2 x 6	475.00	459.60	461.60	474.40	459.40	460.80
50 x 4 x 2	1431.40	1431.00	1421.20	1458.80	1455.00	1444.80
50 x 4 x 4	776.00	758.60	761.00	784.60	769.60	762.80
50 x 4 x 6	592.20	570.80	572.60	594.60	572.60	574.60
50 x 6 x 2	1560.40	1546.20	1528.20	1589.80	1572.20	1566.20
50 x 6 x 4	900.00	889.40	879.60	919.80	915.20	895.20
50 x 6 x 6	708.80	696.80	690.80	718.60	713.60	693.80
50 x 8 x 2	1743.80	1759.00	1709.20	1804.40	1843.40	1764.60
50 x 8 x 4	1023.80	1026.60	1004.00	1049.20	1056.40	1028.80
50 x 8 x 6	789.80	802.60	771.20	812.40	837.60	794.80
100 x 2 x 2	2606.60	2608.60	2601.60	2604.80	2607.00	2601.00
100 x 2 x 4	1356.20	1344.20	1339.80	1353.20	1345.00	1339.40
100 x 2 x 6	915.20	903.80	896.20	916.60	899.20	894.60
100 x 4 x 2	2734.80	2827.00	2723.20	2835.80	2966.80	2794.00
100 x 4 x 4	1440.80	1505.00	1419.20	1452.00	1516.00	1429.40
100 x 4 x 6	1023.40	1034.00	989.60	1061.80	1086.60	1028.00
100 x 6 x 2	2891.40	2989.80	2834.00	3050.00	3189.80	3048.60
100 x 6 x 4	1569.60	1618.40	1532.40	1640.20	1710.00	1614.20
100 x 6 x 6	1140.00	1171.40	1099.60	1177.20	1252.00	1160.20
100 x 8 x 2	3079.20	3172.20	3023.60	3236.40	3336.80	3259.40
100 x 8 x 4	1672.80	1734.20	1645.60	1778.40	1848.00	1796.40
100 x 8 x 6	1277.60	1306.20	1237.80	1315.60	1358.40	1283.80
150 x 2 x 2	3788.00	3841.40	3777.00	3780.40	3816.20	3777.20
150 x 2 x 4	1960.40	1969.40	1943.00	1956.80	1959.20	1942.00
150 x 2 x 6	1297.60	1297.00	1269.80	1302.20	1296.60	1266.60
150 x 4 x 2	4031.40	4145.80	4004.20	4138.00	4287.40	4041.80
150 x 4 x 4	2073.00	2098.80	2007.20	2136.80	2207.40	2052.00
150 x 4 x 6	1424.20	1474.60	1384.60	1485.00	1547.60	1425.60
150 x 6 x 2	4197.20	4339.40	4104.40	4485.60	4603.00	4365.60
150 x 6 x 4	2231.40	2333.00	2195.20	2357.60	2519.00	2289.60
150 x 6 x 6	1610.60	1645.00	1564.20	1657.40	1716.20	1675.00
150 x 8 x 2	4464.40	4561.40	4338.40	4755.40	4976.80	4676.40
150 x 8 x 4	2403.40	2500.40	2364.20	2569.60	2672.00	2544.40
150 x 8 x 6	1724.20	1757.00	1672.40	1821.80	1929.40	1813.00
Average	1483.41	1506.36	1458.24	1529.37	1563.42	1507.54

are shown in Table 5.3.

Based on these results, we make the following observations.

1. Using data set of type I, only in 18 out of 48 instances does GA-Z1 obtain smaller makespan than GA-Z2. However, on the average GA-Z1 gives a smaller average makespan than GA-Z2. A statistical test analysis similar to the one discussed earlier shows that we are not able to reject the null hypothesis at 99% confidence level. In other words, for this type of data set neither GA-Z1 nor GA-Z2 has a clear advantage over the other.

2. Using data set of type II, GA-Z1 gives a smaller average makespan than GA-Z2. In addition, in 41 out of 48 instances, GA-Z1 gives smaller makespan than GA-Z2. A statistical test analysis also supports this finding. We reject the null hypothesis at 99% confidence level.

From these observations, we can conclude that in type I instances neither GA-Z1 nor GA-Z2 has a clear advantage over the other. However, in the type II instances GA-Z1 is more effective than GA-Z2. Therefore, we select $Z1$ as the option employed in this context (i.e., we select algorithm GA-Z1).

5.3.3 Tabu search with vector representation

Similar to the previous algorithms, in the context of tabu search, we have two procedures $Z1$ and $Z2$ to construct a complete schedule for a given permutation

Table 5.3: The makespans obtained using GA-Z1 and GA-Z2.

					Data set of type I		Data set of type II	
N	x	L	x	m_L	GA-Z1	GA-Z2	GA-Z1	GA-Z2
20	x	2	x	2	1175.80	1175.40	550.00	550.00
20	x	2	x	4	504.40	502.60	288.00	287.60
20	x	2	x	6	345.60	345.00	227.00	223.40
20	x	4	x	2	1289.20	1285.60	637.60	637.00
20	x	4	x	4	548.80	546.60	398.00	400.00
20	x	4	x	6	427.40	427.00	316.00	316.60
20	x	6	x	2	1335.80	1322.40	772.20	781.80
20	x	6	x	4	601.00	602.00	495.00	496.00
20	x	6	x	6	525.80	528.80	438.20	439.60
20	x	8	x	2	1284.60	1295.00	915.80	925.80
20	x	8	x	4	647.60	642.60	592.80	603.40
20	x	8	x	6	494.40	490.80	549.20	549.20
50	x	2	x	2	4712.40	4716.80	1326.80	1325.80
50	x	2	x	4	1642.00	1640.00	662.00	665.60
50	x	2	x	6	937.20	939.20	463.20	464.20
50	x	4	x	2	5149.00	5126.00	1409.80	1428.80
50	x	4	x	4	1806.00	1801.20	751.20	758.80
50	x	4	x	6	976.60	974.60	562.20	569.80
50	x	6	x	2	5155.00	5157.00	1506.40	1536.80
50	x	6	x	4	1747.80	1745.40	867.80	886.00
50	x	6	x	6	1121.80	1120.80	675.60	688.80
50	x	8	x	2	4944.20	4994.80	1674.40	1713.80
50	x	8	x	4	1853.60	1842.40	990.20	1014.20
50	x	8	x	6	1064.80	1068.20	753.20	772.80
100	x	2	x	2	15802.40	15796.40	2600.00	2600.60
100	x	2	x	4	4878.40	4876.00	1343.00	1340.20
100	x	2	x	6	2522.60	2519.00	903.00	903.80
100	x	4	x	2	16410.40	16385.00	2714.40	2764.60
100	x	4	x	4	5256.60	5246.80	1414.20	1440.00
100	x	4	x	6	2792.40	2783.80	989.60	1022.40
100	x	6	x	2	17205.60	17231.20	2817.40	2955.40
100	x	6	x	4	5314.80	5338.40	1517.40	1585.60
100	x	6	x	6	2881.40	2879.60	1102.20	1142.00
100	x	8	x	2	17108.60	17207.40	2980.80	3188.60
100	x	8	x	4	5585.40	5577.60	1629.60	1732.80
100	x	8	x	6	2963.60	2966.20	1247.40	1276.40
150	x	2	x	2	35175.60	35189.40	3776.60	3777.80
150	x	2	x	4	9600.20	9594.40	1946.60	1946.80
150	x	2	x	6	4900.00	4898.20	1278.40	1280.20
150	x	4	x	2	35325.00	35361.00	4006.60	4052.80
150	x	4	x	4	10052.00	10070.20	2035.00	2072.80
150	x	4	x	6	5105.20	5113.40	1386.20	1440.20
150	x	6	x	2	36285.20	36244.00	4123.60	4381.60
150	x	6	x	4	10713.40	10711.80	2205.00	2307.40
150	x	6	x	6	5308.60	5293.40	1568.20	1636.00
150	x	8	x	2	36713.60	36544.80	4331.80	4676.20
150	x	8	x	4	10969.20	11164.60	2347.40	2544.80
150	x	8	x	6	5223.00	5246.60	1681.80	1791.20
			Average		7174.67	7177.70	1453.52	1497.83

vector. Thus, in this context, we have two options of algorithms which we refer to as TS-Z1 and TS-Z2, respectively, where TS refers to the tabu search algorithm and Z1 and Z2 refer to the procedure employed. Our objective in this part of experiment is to determine which algorithm is more effective.

In this experiment, we also use the same instances as we used earlier. For each problem instance, we run the programs for 30 seconds of CPU time (omitting the stopping criterion as defined earlier in section 4.4) and report the best makespan obtained. Similar with the experiment using local improvement and genetic algorithm, we use a time limit of 30 seconds of CPU time here for practical reason. The number of iterations performed within this time limit is presented in B. The results of the experiment for data sets of type I and II are shown in Table 5.4.

Based on these results, we make the following observations.

1. Using data set of type I, TS-Z1 gives a smaller average makespan than TS-Z2 and in 25 out of 48 instances TS-Z1 obtains smaller average makespan than TS-Z2. However, a statistical test analysis similar to the one discussed earlier shows that we are not able to reject the null hypothesis at 99% confidence level. In other words, we cannot conclude that either TS-Z1 or TS-Z2 is more effective than the other in this data set.

2. Using data set of type II, TS-Z1 gives a smaller average makespan than TS-Z2 and in 40 out of 48 instances, TS-Z1 obtains smaller makespan than TS-Z2. A statistical test analysis also supports this finding. We reject the

null hypothesis at 99% confidence level.

From these observations, we make a similar conclusion as that for the experiment using genetic algorithm. We conclude that using data set of type I, TS-Z1 and TS-Z2 perform equally good. But, using data set of type II, TS-Z1 is more effective than TS-Z2. Therefore, we select $Z1$ as the option employed in this context (i.e., we select algorithm TS-Z1).

5.3.4 A comparative study of the four algorithms

In this experiment, we compare the four algorithms that we have developed so far (i.e., local improvement with matrix representation, local improvement with vector representation, genetic algorithm with vector representation, and tabu search algorithm with vector representation) with each other. Our objective is to determine the most effective algorithm. Using the same data set as used for the previous experiment, we run the local improvement procedure using matrix representation (LI-M) for 30 seconds of CPU time with as many randomly generated starting points as possible, and retain the best results. As for comparison, we use the results obtained via LI-Z1-L, GA-Z1, and TS-Z1 (from the previous experiment). The results of the experiment for data sets of type I and II are shown in Tables 5.5 and 5.6, respectively.

From these tables, we make the following observations.

1. For the given amount of computation time, vector representation performs

Table 5.4: The makespans obtained using TS-Z1 and TS-Z2.

					Data set of type I		Data set of type II	
N	x	L	x	m_L	TS-Z1	TS-Z2	TS-Z1	TS-Z2
20	x	2	x	2	1176.20	1174.60	550.00	550.00
20	x	2	x	4	500.20	500.20	280.20	280.60
20	x	2	x	6	344.20	340.60	220.20	220.60
20	x	4	x	2	1284.00	1281.80	631.80	634.20
20	x	4	x	4	540.20	542.60	394.60	395.60
20	x	4	x	6	415.40	415.40	312.20	312.80
20	x	6	x	2	1316.40	1328.80	765.20	772.00
20	x	6	x	4	585.00	591.80	487.40	488.00
20	x	6	x	6	502.40	502.60	436.80	437.20
20	x	8	x	2	1285.00	1274.00	904.40	913.40
20	x	8	x	4	626.40	626.80	589.80	590.20
20	x	8	x	6	478.40	480.80	543.80	544.80
50	x	2	x	2	4697.40	4698.80	1325.00	1325.60
50	x	2	x	4	1641.20	1642.60	656.00	655.60
50	x	2	x	6	942.00	936.40	449.80	449.80
50	x	4	x	2	5122.40	5099.20	1401.20	1409.80
50	x	4	x	4	1815.20	1807.60	731.40	740.00
50	x	4	x	6	977.00	983.80	546.40	544.60
50	x	6	x	2	5149.80	5152.20	1492.80	1517.60
50	x	6	x	4	1755.00	1755.00	848.60	858.00
50	x	6	x	6	1132.60	1138.40	659.20	666.20
50	x	8	x	2	5020.60	4985.00	1663.20	1708.20
50	x	8	x	4	1873.20	1870.20	977.00	993.00
50	x	8	x	6	1072.00	1074.20	747.20	779.40
100	x	2	x	2	15770.60	15757.80	2600.20	2599.20
100	x	2	x	4	4880.20	4883.80	1338.00	1337.00
100	x	2	x	6	2523.00	2530.00	888.00	892.80
100	x	4	x	2	16421.40	16379.00	2705.40	2744.20
100	x	4	x	4	5322.80	5298.40	1399.00	1427.60
100	x	4	x	6	2813.00	2816.80	984.60	1021.60
100	x	6	x	2	17231.20	17299.60	2816.00	2954.60
100	x	6	x	4	5469.40	5471.20	1538.20	1611.00
100	x	6	x	6	2926.40	2929.40	1099.00	1142.40
100	x	8	x	2	17227.40	17186.60	3044.80	3222.60
100	x	8	x	4	5787.20	5706.00	1643.20	1752.80
100	x	8	x	6	3091.80	3079.00	1244.80	1291.60
150	x	2	x	2	35099.20	35104.40	3776.20	3776.40
150	x	2	x	4	9632.60	9628.20	1942.60	1942.00
150	x	2	x	6	4922.20	4911.80	1270.60	1269.80
150	x	4	x	2	35404.00	35469.40	3999.40	4049.80
150	x	4	x	4	10138.00	10163.40	2023.40	2063.80
150	x	4	x	6	5178.40	5151.40	1406.20	1424.00
150	x	6	x	2	36328.20	36318.60	4202.00	4471.60
150	x	6	x	4	10831.80	10898.00	2235.40	2342.20
150	x	6	x	6	5369.40	5375.20	1595.60	1665.20
150	x	8	x	2	36714.60	36818.40	4404.80	4747.40
150	x	8	x	4	11159.60	11165.60	2409.40	2570.60
150	x	8	x	6	5327.20	5301.40	1710.40	1820.20
			Average		7204.62	7205.14	1456.07	1498.49

better than matrix representation. The resulting makespans obtained using LI-Z1-L, GA-Z1, and TS-Z1 are lower than those obtained using matrix representation (LI-M) in every instance considered in the experiment. To further investigate algorithm LI-M, we extend our experiment by running the program for 90 seconds of CPU time. We observe a similar result.

2. Comparison of the results obtained using LI-Z1-L, GA-Z1, and TS-Z1 with each other shows that the average makespans obtained using GA-Z1 is smaller than those obtained using either LI-Z1-L or TS-Z1 in both types of data set.

3. Using data set of type I, we determine that GA-Z1 yields a smaller average makespan than both LI-Z1-L and TS-Z1. A comparison on the makespan of each problem size shows that in 46 out of 48 instances, GA-Z1 gives smaller makespan than LI-Z1-L, and in 32 out of 48 instances GA-Z1 gives smaller makespan than TS-Z1. Comparing the results obtained using TS-Z1 with LI-Z1-L, we can see that in 37 out of 48 instances, TS-Z1 yields smaller makespan than LI-Z1-L. The Wilcoxon signed rank test shows that we reject the null hypothesis at 95% confidence level for the following pairs of algorithms: GA-Z1 vs TS-Z1, GA-Z1 vs LI-Z1-L, and TS-Z1 vs LI-Z1-L.

4. Using data set of type II, we also determine that GA-Z1 yields a smaller average makespan than both LI-Z1-L and TS-Z1. Comparing the makespan obtained, in 36 out of 48 instances TS-Z1 gives smaller makespan than GA-

Z1, and in 35 out of 48 instances TS-Z1 gives smaller makespan than LI-Z1-L. A statistical test analysis shows that we reject the null hypothesis at 95% confidence level for the pairs of algorithms GA-Z1 vs LI-Z1-L. On the other hand, we are not able to reject the null hypothesis for the following pairs of algorithms at 95% confidence level: TS-Z1 vs GA-Z1 and TS-Z1 vs LI-Z1-L.

To summarize, we make the following conclusions.

1. Among the four different search algorithms that we have designed, algorithms LI-Z1-L, GA-Z1, and TS-Z1, which are based on the vector representation, perform more effectively than algorithm LI-M, which is based on the matrix representation.

2. Comparing the three algorithms which are based on vector representation, we conclude that using data set of type I, GA-Z1 performs better than both LI-Z1-L and TS-Z1, and TS-Z1 performs better than LI-Z1-L. However, using data set of type II, algorithms GA-Z1 and TS-Z1 perform equally good and are better than algorithm LI-Z1-L.

Based on these results, we select algorithms GA-Z1 and TS-Z1 for a more thorough investigation, since these two algorithms seem to be the most effective among the four proposed algorithms considered.

Table 5.5: The makespans obtained using LI-M, LI-Z1-L, GA-Z1 and TS-Z1 (data set of type I).

N x L x m_L	LI-M	LI-Z1-L	GA-Z1	TS-Z1
20 x 2 x 2	1205.80	1191.60	1175.80	1176.20
20 x 2 x 4	515.60	506.80	504.40	500.20
20 x 2 x 6	356.40	350.20	345.60	344.20
20 x 4 x 2	1365.80	1316.60	1289.20	1284.00
20 x 4 x 4	575.60	557.80	548.80	540.20
20 x 4 x 6	433.00	429.20	427.40	415.40
20 x 6 x 2	1457.20	1384.40	1335.80	1316.40
20 x 6 x 4	625.00	603.80	601.00	585.00
20 x 6 x 6	545.00	527.80	525.80	502.40
20 x 8 x 2	1433.00	1334.80	1284.60	1285.00
20 x 8 x 4	676.60	652.40	647.60	626.40
20 x 8 x 6	503.40	493.40	494.40	478.40
50 x 2 x 2	4819.20	4750.60	4712.40	4697.40
50 x 2 x 4	1684.60	1660.00	1642.00	1641.20
50 x 2 x 6	966.00	948.80	937.20	942.00
50 x 4 x 2	5424.20	5237.60	5149.00	5122.40
50 x 4 x 4	1927.80	1843.20	1806.00	1815.20
50 x 4 x 6	1043.00	996.40	976.60	977.00
50 x 6 x 2	5701.80	5317.20	5155.00	5149.80
50 x 6 x 4	1902.60	1783.00	1747.80	1755.00
50 x 6 x 6	1219.80	1149.20	1121.80	1132.60
50 x 8 x 2	5655.20	5194.80	4944.20	5020.60
50 x 8 x 4	2101.60	1909.20	1853.60	1873.20
50 x 8 x 6	1177.20	1089.40	1064.80	1072.00
100 x 2 x 2	15971.20	15850.00	15802.40	15770.60
100 x 2 x 4	4971.40	4911.00	4878.40	4880.20
100 x 2 x 6	2577.00	2545.20	2522.60	2523.00
100 x 4 x 2	17032.00	16537.60	16410.40	16421.40
100 x 4 x 4	5515.80	5340.80	5256.60	5322.80
100 x 4 x 6	2947.40	2841.00	2792.40	2813.00
100 x 6 x 2	18119.00	17381.00	17205.60	17231.20
100 x 6 x 4	5746.60	5432.60	5314.80	5469.40
100 x 6 x 6	3139.60	2947.20	2881.40	2926.40
100 x 8 x 2	18256.40	17181.00	17108.60	17227.40
100 x 8 x 4	6167.80	5700.20	5585.40	5787.20
100 x 8 x 6	3351.80	3043.20	2963.60	3091.80
150 x 2 x 2	35441.80	35207.60	35175.60	35099.20
150 x 2 x 4	9736.60	9655.40	9600.20	9632.60
150 x 2 x 6	4992.40	4934.60	4900.00	4922.20
150 x 4 x 2	36094.00	35427.00	35325.00	35404.00
150 x 4 x 4	10376.00	10137.20	10052.00	10138.00
150 x 4 x 6	5323.40	5173.00	5105.20	5178.40
150 x 6 x 2	37287.60	36322.60	36285.20	36328.20
150 x 6 x 4	11229.20	10806.60	10713.40	10831.80
150 x 6 x 6	5599.60	5379.00	5308.60	5369.40
150 x 8 x 2	37813.00	36596.60	36713.60	36714.60
150 x 8 x 4	11716.20	11112.20	10969.20	11159.60
150 x 8 x 6	5629.00	5294.80	5223.00	5327.20
Average	7465.63	7228.87	7174.67	7204.62

Table 5.6: The makespans obtained using LI-M, LI-Z1-L, GA-Z1 and TS-Z1 (data set of type II).

N x L x m_L	LI-M	LI-Z1-L	GA-Z1	TS-Z1
20 x 2 x 2	552.60	550.00	550.00	550.00
20 x 2 x 4	306.60	287.80	288.00	280.20
20 x 2 x 6	238.40	225.00	227.00	220.20
20 x 4 x 2	678.20	640.60	637.60	631.80
20 x 4 x 4	433.00	407.00	398.00	394.60
20 x 4 x 6	337.60	318.80	316.00	312.20
20 x 6 x 2	837.60	785.40	772.20	765.20
20 x 6 x 4	539.60	502.20	495.00	487.40
20 x 6 x 6	452.00	439.20	438.20	436.80
20 x 8 x 2	995.20	934.80	915.80	904.40
20 x 8 x 4	645.80	601.20	592.80	589.80
20 x 8 x 6	575.40	550.00	549.20	543.80
50 x 2 x 2	1342.80	1325.20	1326.80	1325.00
50 x 2 x 4	686.20	661.60	662.00	656.00
50 x 2 x 6	495.20	461.60	463.20	449.80
50 x 4 x 2	1531.40	1421.20	1409.80	1401.20
50 x 4 x 4	817.60	761.00	751.20	731.40
50 x 4 x 6	625.80	572.60	562.20	546.40
50 x 6 x 2	1706.00	1528.20	1506.40	1492.80
50 x 6 x 4	972.00	879.60	867.80	848.60
50 x 6 x 6	753.00	690.80	675.60	659.20
50 x 8 x 2	1879.80	1709.20	1674.40	1663.20
50 x 8 x 4	1089.40	1004.00	990.20	977.00
50 x 8 x 6	835.80	771.20	753.20	747.20
100 x 2 x 2	2657.40	2601.60	2600.00	2600.20
100 x 2 x 4	1384.60	1339.80	1343.00	1338.00
100 x 2 x 6	945.00	896.20	903.00	888.00
100 x 4 x 2	2954.20	2723.20	2714.40	2705.40
100 x 4 x 4	1525.00	1419.20	1414.20	1399.00
100 x 4 x 6	1067.00	989.60	989.60	984.60
100 x 6 x 2	3119.00	2834.00	2817.40	2816.00
100 x 6 x 4	1670.80	1532.40	1517.40	1538.20
100 x 6 x 6	1201.80	1099.60	1102.20	1099.00
100 x 8 x 2	3355.40	3023.60	2980.80	3044.80
100 x 8 x 4	1771.80	1645.60	1629.60	1643.20
100 x 8 x 6	1342.40	1237.80	1247.40	1244.80
150 x 2 x 2	3918.80	3777.00	3776.60	3776.20
150 x 2 x 4	1999.60	1943.00	1946.60	1942.60
150 x 2 x 6	1331.20	1269.80	1278.40	1270.60
150 x 4 x 2	4262.20	4004.20	4006.60	3999.40
150 x 4 x 4	2169.20	2007.20	2035.00	2023.40
150 x 4 x 6	1512.60	1384.60	1386.20	1406.20
150 x 6 x 2	4522.20	4104.40	4123.60	4202.00
150 x 6 x 4	2367.40	2195.20	2205.00	2235.40
150 x 6 x 6	1671.60	1564.20	1568.20	1595.60
150 x 8 x 2	4660.60	4338.40	4331.80	4404.80
150 x 8 x 4	2549.20	2364.20	2347.40	2409.40
150 x 8 x 6	1800.80	1672.40	1681.80	1710.40
Average	1564.31	1458.24	1453.52	1456.07

5.3.5 A further comparative study of the algorithms

From the previous experiment, we can see that the proposed genetic algorithm (GA-Z1) and tabu search procedure (TS-Z1) perform better than the proposed local improvement procedure (LI-Z1-L) in the context of vector representation. Using data set of type I, our previous experiment have shown that GA-Z1 performs better than TS-Z1. However, using data set of type II we are not able to conclude that either GA-Z1 or TS-Z1 is better than the other.

In this part of the experiment, our objective is to determine if either one of these two algorithms actually performs better than the other by further investigating their performance. For this purpose, we run each program until the specified termination criterion (as defined in section 4.3 and 4.4) is met and retain the best result. The results of the experiment are shown in Table 5.7 and the corresponding CPU times are shown in Table 5.8.

Based on these results, we make the following observations.

- Using data set of type I, we observe that in 26 out of 48 instances GA-Z1 performs better than TS-Z1. However, TS-Z1 gives smaller average makespan than GA-Z1. A statistical test analysis similar to the one discussed earlier shows that we are not able to reject the null hypothesis at 99% confidence level. Observing the running time, TS-Z1 requires smaller average CPU time (55.16 seconds) than GA-Z1 (83.4 seconds). Therefore, we conclude that in this context, TS-Z1 is more favorable than GA-Z1.

Notice that from the previous experiment (for a given amount of CPU time, i.e., 30 seconds), GA-Z1 performs better than TS-Z1. This indicates that GA-Z1 obtains better makespan than TS-Z1 at the beginning of the iteration. However, as these algorithms are allowed to run longer, GA-Z1 does not perform as effective as TS-Z1 in exploring the solution space to obtain better solutions. (Observe that in this part of experiment, in almost all instances GA-Z1 requires longer CPU time than TS-Z1).

- Using data set of type II, TS-Z1 yields smaller average makespans than GA-Z1. A statistical test analysis also supports this finding. We reject the null hypothesis at 99% confidence level. In addition, in 43 out of 48 instances, TS-Z1 gives smaller makespan than GA-Z1. Observation on the running time shows that TS-Z1 requires smaller average CPU time (53.98 seconds) than GA-Z1 (68.53 seconds).

From the above observations, we conclude that among the proposed algorithms, TS-Z1 is the most effective algorithm in the context of FSPM problem (although using data set of type I, GA-Z1 also gives results close to those obtained using TS-Z1). As for the next experiment, we investigate the quality of the solutions obtained using TS-Z1 as compared with some reference values.

Table 5.7: The makespans obtained using GA-Z1 and TS-Z1.

					Data set of type I		Data set of type II	
N	x	L	x	m_L	GA-Z1	TS-Z1	GA-Z1	TS-Z1
20	x	2	x	2	1182.00	1184.20	551.00	550.60
20	x	2	x	4	504.60	505.60	288.00	282.20
20	x	2	x	6	347.80	350.60	226.40	222.60
20	x	4	x	2	1293.40	1299.60	638.40	639.80
20	x	4	x	4	553.60	554.80	404.20	397.20
20	x	4	x	6	429.60	430.80	316.80	315.00
20	x	6	x	2	1342.40	1345.20	781.20	773.80
20	x	6	x	4	601.60	599.80	496.80	492.60
20	x	6	x	6	527.40	535.60	442.80	439.40
20	x	8	x	2	1297.60	1302.00	915.20	917.80
20	x	8	x	4	646.80	642.40	593.00	593.20
20	x	8	x	6	491.60	501.80	547.00	545.00
50	x	2	x	2	4735.60	4718.80	1326.40	1325.60
50	x	2	x	4	1650.20	1653.40	661.40	659.80
50	x	2	x	6	941.80	942.20	463.80	452.20
50	x	4	x	2	5156.60	5104.20	1413.80	1403.40
50	x	4	x	4	1807.00	1805.20	747.20	731.80
50	x	4	x	6	979.60	992.60	565.40	543.60
50	x	6	x	2	5160.80	5113.60	1508.60	1505.20
50	x	6	x	4	1735.80	1747.40	866.60	848.00
50	x	6	x	6	1119.00	1137.40	674.60	651.80
50	x	8	x	2	4960.40	4928.80	1669.00	1639.80
50	x	8	x	4	1842.00	1847.60	984.00	961.00
50	x	8	x	6	1066.00	1060.00	747.80	729.40
100	x	2	x	2	15810.40	15748.00	2601.00	2605.00
100	x	2	x	4	4874.40	4881.60	1345.00	1339.20
100	x	2	x	6	2517.00	2523.60	900.80	890.40
100	x	4	x	2	16356.00	16317.20	2720.00	2696.60
100	x	4	x	4	5230.40	5220.00	1397.60	1391.80
100	x	4	x	6	2766.80	2787.40	983.20	977.60
100	x	6	x	2	16947.00	16905.40	2810.20	2794.40
100	x	6	x	4	5309.00	5257.40	1502.00	1480.40
100	x	6	x	6	2845.80	2855.40	1086.80	1045.00
100	x	8	x	2	16873.80	16522.20	2970.40	2974.40
100	x	8	x	4	5431.20	5437.40	1605.20	1577.80
100	x	8	x	6	2893.20	2901.60	1215.00	1179.60
150	x	2	x	2	35144.40	35067.00	3780.40	3778.20
150	x	2	x	4	9603.00	9600.60	1948.60	1945.40
150	x	2	x	6	4891.00	4898.60	1274.60	1264.00
150	x	4	x	2	35183.80	34956.80	4002.20	3986.60
150	x	4	x	4	9984.00	9948.00	2016.20	1989.60
150	x	4	x	6	5039.40	5053.00	1382.00	1356.80
150	x	6	x	2	35700.40	35313.20	4060.60	4051.80
150	x	6	x	4	10445.80	10481.20	2147.20	2122.80
150	x	6	x	6	5155.00	5205.60	1532.60	1501.40
150	x	8	x	2	35885.20	35304.00	4231.80	4216.60
150	x	8	x	4	10720.60	10651.80	2294.20	2258.60
150	x	8	x	6	5010.20	5002.20	1645.20	1595.80
				Average	7103.98	7065.48	1443.38	1430.01

Table 5.8: The CPU time (seconds) obtained using GA-Z1 and TS-Z1

	Data set of type I		Data set of type II	
N x L x m_L	GA-Z1	TS-Z1	GA-Z1	TS-Z1
20 x 2 x 2	3.34	0.52	1.69	0.33
20 x 2 x 4	2.66	0.63	2.73	0.63
20 x 2 x 6	3.98	0.67	3.05	0.69
20 x 4 x 2	5.19	1.12	4.94	1.09
20 x 4 x 4	5.49	1.48	6.24	1.69
20 x 4 x 6	5.54	1.38	6.92	1.52
20 x 6 x 2	6.81	1.32	8.16	1.68
20 x 6 x 4	9.48	2.38	9.97	1.83
20 x 6 x 6	7.83	2.31	5.95	1.82
20 x 8 x 2	9.19	2.01	11.39	1.55
20 x 8 x 4	12.83	3.24	13.06	3.23
20 x 8 x 6	10.45	2.65	15.10	2.79
50 x 2 x 2	7.80	5.16	7.14	2.71
50 x 2 x 4	9.72	7.69	8.03	4.50
50 x 2 x 6	12.76	6.56	13.71	7.38
50 x 4 x 2	22.74	13.29	14.91	12.37
50 x 4 x 4	19.38	20.29	22.86	14.84
50 x 4 x 6	22.08	11.91	33.91	15.16
50 x 6 x 2	28.06	20.49	26.22	12.08
50 x 6 x 4	38.26	21.32	47.94	19.77
50 x 6 x 6	43.89	29.26	43.95	29.23
50 x 8 x 2	56.20	37.18	45.03	33.90
50 x 8 x 4	62.36	32.04	54.13	30.65
50 x 8 x 6	36.63	39.74	60.59	46.64
100 x 2 x 2	24.52	20.39	12.39	6.41
100 x 2 x 4	24.51	16.91	22.57	13.54
100 x 2 x 6	29.20	21.52	39.48	22.08
100 x 4 x 2	61.65	49.47	42.98	40.23
100 x 4 x 4	75.16	59.35	88.74	44.23
100 x 4 x 6	85.22	58.65	106.67	56.89
100 x 6 x 2	130.25	72.33	82.03	58.31
100 x 6 x 4	100.01	90.82	97.59	79.91
100 x 6 x 6	121.70	72.40	116.65	101.70
100 x 8 x 2	122.65	101.08	145.23	49.53
100 x 8 x 4	141.94	115.41	111.67	118.97
100 x 8 x 6	181.33	141.09	139.28	155.57
150 x 2 x 2	48.36	26.43	23.20	16.41
150 x 2 x 4	53.74	35.16	38.36	18.08
150 x 2 x 6	62.31	33.98	42.96	37.04
150 x 4 x 2	231.49	94.26	81.99	46.01
150 x 4 x 4	160.52	117.24	107.45	74.30
150 x 4 x 6	182.72	97.15	100.69	86.44
150 x 6 x 2	217.39	139.57	165.24	97.31
150 x 6 x 4	267.82	144.98	243.92	185.85
150 x 6 x 6	322.96	124.21	235.24	216.01
150 x 8 x 2	269.92	214.67	246.39	176.41
150 x 8 x 4	273.59	244.28	252.59	258.90
150 x 8 x 6	371.62	291.45	278.60	382.74
Average	83.40	55.16	68.53	53.98

5.4 An empirical study of the quality of solutions

In the experiment that we reported so far, we have focused on comparing the proposed procedures with each other, with the expressed objective of determining which procedure is more effective than the others. In other words, we have so far focused on a relative comparison of these procedures with each other. We now extend our experiment by comparing the results obtained via the best algorithm (i.e., TS-Z1) with some corresponding reference values in order to evaluate the quality of the solutions. For this purpose, we use the results obtained from the previous experiment as shown in section 5.3.5 (i.e., upon running the program until the specified termination criterion is met).

In subsection 5.4.1, we present the method of evaluating the solution quality for each data type. In subsection 5.4.2, we present the experimental results using data set of type I. The results of the experiment using data set of type II are presented in subsection 5.4.3. Here, we also compare the results of algorithm TS-Z1 with the results of Nowicki et al. [44].

5.4.1 Performance evaluation

The following discussion presents the methods that we employed to evaluate the quality of the solutions obtained for instances for each type of data set.

Performance evaluation of the data set of type I

Since we know the optimal makespan for each instance of the problem in this group, we compare the results of the algorithms with the corresponding optimal makespan. The evaluation is based on the deviation of the makespan obtained using the proposed algorithms from the optimal makespan, and it is computed as follows.

$$\delta_h = \frac{C_h - C_{opt}}{C_{opt}}$$

where

δ_h = The performance ratio of algorithm h

C_h = The makespan obtained using the proposed algorithm h

C_{opt} = The optimal makespan

Naturally, $\delta_h \geq 0$ for all algorithms in all instances, and $\delta_h = 0$ implies that the proposed algorithm has obtained the optimal solution.

Performance evaluation of the data set of type II

For this class of instances, the corresponding optimal makespans are not known. The optimal makespan of each instance can actually be obtained using branch-and-bound method or using total enumeration. However, these methods become inefficient for $N > 8$, as discussed in [18] and [57]. Instead, in this group of instances, we employ two other reference values. One is the best makespan obtained

via other heuristic procedures from the open literatures, and the other is a lower bound on the optimal makespan of the corresponding instance.

As for using the best makespan obtained for each instance using other procedures as a reference value, we employ three known heuristic procedures from the open literatures. The evaluation of the solution quality is based on the relative deviation of the makespan obtained using the proposed algorithms with respect to the reference makespan. For this purpose, we use the best makespan obtained using heuristics $DK1$, $DK2$, and $DK3$ as discussed in [18], and the relative performance of each procedure is measured as follow.

$$\upsilon_h = \frac{C_{Ref} - C_h}{C_{Ref}}$$

where

υ_h = The relative performance ratio of algorithm h

C_h = The makespan obtained using proposed algorithm h

C_{Ref} = The best makespan obtained using $DK1$, $DK2$, and $DK3$

Here, $\upsilon_h > 0$ indicates that the makespan obtained by the proposed algorithm is smaller than the reference makespan.

In comparing the experimental results with the lower bound on the optimal makespan, the evaluation is based on the deviation of the makespan obtained using the proposed algorithms from the lower bound. The deviation is computed using the following formulation.

$$\eta_h = \frac{C_h - LB}{LB}$$

where

η_h = The performance ratio of algorithm h

C_h = The makespan obtained using proposed algorithm h

LB = The best lower bound

As for the lower bound, we employ the lower bound as discussed in section 5.2.

5.4.2 Results of experiment using data set of type I

In this experiment, our objective is to measure the performance of TS-Z1 in terms of the solution quality. As we have mentioned earlier, using this type of data set, the performance measure is based on the deviation of the resulting makespan from the optimal solution. The results of the experiment are shown in Table 5.9. In this table, the first column indicates the size of problem instances and the remaining columns indicate the performance ratio corresponding to the makespans obtained via each procedure. As for comparison, in this table we also present the performance ratio corresponding to the makespans obtained using heuristics $H5$ [58], LV [32], and DK, where DK represents the best results obtained using heuristic procedure $DK1$, $DK2$, or $DK3$ [18] for a given instance. (Notice that $H5$ and LV apply only to problem instances with $L = 2$).

From this table, we make the following observations.

- On the average, TS-Z1 gives a relative performance ratio of 0.0454, i.e., the makespan obtained using TS-Z1 is on average 4.54% above the optimal

makespan. For this type of instances with $L = 2$, $H5$ with an average performance ratio of 0.0462 performs better than LV with the average performance ratio of 0.0632. However, both procedures perform worse than TS-Z1 with an average performance ratio of 0.0189 (for $L = 2$).

- DK gives smaller average performance ratio than TS-Z1. In addition, in two instances DK obtains the optimal makespan. Our observation shows that this is primarily due the structure of these type of instances, which is conducive to the pattern by which DK (particularly $DK1$) constructs a complete schedule. On the other hand, TS-Z1 does not take advantage of this structure in any way. Indeed, we believe that this type of problem instances are among the most difficult instances for any search procedure that does not take advantage of the special structure of the data set (such as TS-Z1), since the problem has a small number of good solutions.

- For a given number of jobs (N) and number machines (m_L), TS-Z1 gives better performance (lower δ_h) at a smaller number of stages (L).

- A similar pattern also occurs for a given number of jobs (N) and number of stages (L). TS-Z1 gives better performance (lower δ_h) at a smaller number of machines (m_L).

From these observations, we conclude that the results obtained via TS-Z1 are better than those obtained via constructive procedures $H5$ and LV (for $L = 2$),

but they are not always better than those obtained via procedure DK. On the average, TS-Z1 yields an average performance ratio of 4.54% above the optimal makespan.

5.4.3 Results of experiment using data set of type II

Since the optimal makespan for instances in this category are not known, we use the lower bound and the reference makespans for evaluating the results as discussed in section 5.4.1. As part of the objective of this experiment, we also intent to determine the solution quality of algorithm TS-Z1 as compared with another search algorithm from open literature. For this purpose, we use the results of Nowicki et al. [44] which were obtained using tabu search with matrix representation.

As we mentioned in section 5.1, we also conduct an experiment using data set of type II in which the number of machines at each stage is generated randomly, i.e., m_ℓ is generated randomly following a uniform distribution between 1 and 5 ($P(m_\ell = k) = \frac{1}{5}$, for $k = 1, ..., 5$, and for all ℓ). The experimental results are presented in Tables 5.10 to 5.12.

Table 5.10 shows the relative performance ratio (υ_h) of the makespans obtained using TS-Z1 with respect to the associated makespans obtained by other heuristic procedures. As for comparison, in this table we also present the relative performance ratio of the makespan obtained using procedures $H5$ and LV with respect to the reference makespan for problem instances with $L = 2$. The last two columns

Table 5.9: Deviation (δ_h) from the optimal makespan (data set of type I).

N x L x m_L	TS-Z1	DK	H5	LV
20 x 2 x 2	0.0160	0.0238	0.0376	0.0392
20 x 2 x 4	0.0239	0.0654	0.0892	0.1243
20 x 2 x 6	0.0341	0.0018	0.0777	0.1612
20 x 4 x 2	0.0308	0.0407		
20 x 4 x 4	0.0453	0.0022		
20 x 4 x 6	0.0705	0.0011		
20 x 6 x 2	0.0352	0.0052		
20 x 6 x 4	0.0462	0.0048		
20 x 6 x 6	0.0767	0.0000		
20 x 8 x 2	0.0449	0.0031		
20 x 8 x 4	0.0471	0.0054		
20 x 8 x 6	0.0899	0.0000		
50 x 2 x 2	0.0122	0.0080	0.0285	0.0264
50 x 2 x 4	0.0238	0.0202	0.0648	0.0749
50 x 2 x 6	0.0258	0.0417	0.0840	0.1199
50 x 4 x 2	0.0243	0.0144		
50 x 4 x 4	0.0445	0.0556		
50 x 4 x 6	0.0645	0.0796		
50 x 6 x 2	0.0339	0.0183		
50 x 6 x 4	0.0623	0.0637		
50 x 6 x 6	0.0714	0.0534		
50 x 8 x 2	0.0467	0.0195		
50 x 8 x 4	0.0639	0.0483		
50 x 8 x 6	0.0631	0.0448		
100 x 2 x 2	0.0090	0.0026	0.0169	0.0146
100 x 2 x 4	0.0173	0.0106	0.0349	0.0421
100 x 2 x 6	0.0215	0.0271	0.0579	0.0721
100 x 4 x 2	0.0295	0.0063		
100 x 4 x 4	0.0390	0.0293		
100 x 4 x 6	0.0534	0.0485		
100 x 6 x 2	0.0389	0.0077		
100 x 6 x 4	0.0594	0.0388		
100 x 6 x 6	0.0702	0.0678		
100 x 8 x 2	0.0490	0.0079		
100 x 8 x 4	0.0759	0.0325		
100 x 8 x 6	0.0798	0.0547		
150 x 2 x 2	0.0088	0.0033	0.0091	0.0085
150 x 2 x 4	0.0149	0.0093	0.0238	0.0261
150 x 2 x 6	0.0194	0.0113	0.0302	0.0493
150 x 4 x 2	0.0229	0.0048		
150 x 4 x 4	0.0373	0.0267		
150 x 4 x 6	0.0480	0.0336		
150 x 6 x 2	0.0346	0.0055		
150 x 6 x 4	0.0603	0.0240		
150 x 6 x 6	0.0772	0.0446		
150 x 8 x 2	0.0482	0.0110		
150 x 8 x 4	0.0833	0.0243		
150 x 8 x 6	0.0848	0.0444		
Average	0.0454	0.0250	0.0462	0.0632

of this table show the performance ratio values as a result of study by Nowicki et al. (as shown in [44]). In these two columns, v^A indicates the best results obtained during the first 1,000 iteration, while v^B indicates the best results obtained during the first 10,000 iteration. Since we do not have access to the code of the algorithm used by Nowicki et al., we quote these values from their paper. In comparing their results with those obtained using our proposed algorithm, we must consider the following issues:

1. Since we do not have access to the exact same instances as they used and the resulting makespans, a direct comparison is not feasible for us at this time. In the experiment, we use the same method in generating the instances and we generate the same size of instances as they do (see page 55). As such, we expect that on average the relative performance ratios are comparable.

2. In comparing with other procedures, Nowicki et al. use the best of 6 different heuristics procedure from open literature as the reference values, i.e., the best of heuristics $DK1$, $DK2$, $DK3$ [18], W [66], S [56], and HS [26]. In our experiment, we use the best of only 3 heuristics $DK1$, $DK2$, and $DK3$. Since it is conceivable that in some instances either heuristic W or S or HS might obtain a better value as compared with $DK1$, $DK2$, and $DK3$, it follows that our performance measures might be somewhat higher than they would have been if we also employed algorithms W, S, and HS in our experiment.

The observations that we made from Table 5.10 are as follow. (Note that in this

table, a negative value for the performance ratio indicates that the corresponding makespan is indeed larger than the reference makespan).

- In all instances, TS-Z1 obtains smaller makespan than the reference values with an average relative performance ratio of 5.46% below the reference makespan.

- For $L = 2$, TS-Z1 gives an average performance ratio of 0.0236 which is better (higher) than the average performance of $H5$ and LV with the average performance ratio of 0.0162 and -0.0146, respectively.

- In 33 out of 48 instances, TS-Z1 gives higher relative performance ratio than υ^A and in 23 out of 48 instances, TS-Z1 gives higher relative performance than υ^B. On the average, the relative performance ratio TS-Z1 is better than υ^A and slightly lower than υ^B. The Wilcoxon signed rank test shows that we are not able to reject the null hypothesis at 99% confidence level for the results obtained using TS-Z1 and υ^B. Comparing the results obtained using TS-Z1 with υ^A, a similar test shows that we reject the null hypothesis at 99% confidence level. In other words, there is a strong statistical evidence that the results obtained via TS-Z1 are better than those obtained by Nowicki et al. after 1,000 iterations.

  (As we mentioned earlier, Nowicki et al. use more heuristics as the reference values than we do, namely heuristic W [66], S [56], and HS [26]. If we

assume that $DK1$, $DK2$, and $DK3$ perform better than W, S, and HS for the given instances, we may conclude that based on the above observations, TS-Z1 performs better than the algorithm by Nowicki et al. However, if heuristics W, S, and HS perform better than $DK1$, $DK2$, and $DK3$ for the given instances, then the performance ratio of TS-Z1 will actually decrease).

The relative performance ratio (η_h) of the makespans obtained using TS-Z1 with respect to the associated lower bound for the cases of identical and non-identical number of machines are shown in Tables 5.11 and 5.12, respectively. In Table 5.12, we also present the corresponding CPU time (seconds) for TS-Z1. (Notice that the corresponding CPU times for the case of identical number of machines at all stages are shown in Table 5.8).

As for comparison, in these tables we also present the relative performance ratio obtained using heuristics $H5$, LV, and DK. In the last two columns of these tables, we present the performance ratio with respect to the lower bound obtained by Nowicki et al. for the corresponding problem sizes. These values are presented in [44]. Here, η^A and η^B indicate the performance ratio values as a result of study by Nowicki et al. during the first 1,000 and 10,000 iteration, respectively.

From these tables, we make the following observations.

- For the case of identical number of machines at all stages (Table 5.11), TS-Z1 obtains an average performance ratio of 4.32% above the lower bound. This values is better (lower) than the average performance ratio of Nowicki et al.

during the first 1,000 (η^A) iteration, but higher than the corresponding result during the first 10,000 iteration (η^B). This observation is consistent with our previous observation with reference makespan.

- For the case of non-identical number of machines at different stages (Table 5.12), TS-Z1 obtains an average performance ratio of 1.00% above the lower bound. This value is smaller than the values obtained by Nowicki et al.

  (Again, notice that Nowicki et al. use the lower bound to the problem which is obtained by introducing job preemption and the relaxation of machine capacities [31]; as a result, the numbers are not directly comparable.)

- For $L = 2$, TS-Z1 gives an average performance ratio of 1.23% above the lower bound for the case of identical number of machines at all stages. The corresponding value for the case of non-identical number of machines at different stages is 0.23%. Compared with the results obtained using constructive procedures $H5$, LV, and DK, TS-Z1 gives a lower average performance ratio, although its corresponding CPU time requirement are relatively larger.

- For the case of identical number of machines at all stages (Table 5.11), the performance ratio tends to increase as the number of stages increase and as the number of jobs increase. The number of machines at each stage (m_ℓ) does not seem to have an effect that follows a particular pattern on the performance ratio.

- For the case of non-identical number of machines (Table 5.12), we do not observe a particular pattern on the performance ratio due to the variation on the number of jobs and the number of stages.

From the observations of Tables 5.10 and 5.11, we can see that the values of the performance ratio obtained using our proposed algorithm (TS-Z1) do not differ significantly from those obtained by Nowicki et al., and from Table 5.12 we observe that the average performance ratio obtained using TS-Z1 is very close to the lower bound.

Although we use different reference values to measure the performance of our proposed algorithm as compared with Nowicki et al. (in terms of the method of obtaining the reference values and the lower bound used), we conclude that based on this comparison, our proposed algorithm TS-Z1 performs reasonably well as compared with that of Nowicki et al. But, the solution representation (vector representation) that we used is significantly simpler than that used by Nowicki et al., and it is readily extendable to the problem with limited buffer sizes. As such, based on this outcome, we extend our study to the FSPM/b problem using vector representation as will be discussed in the subsequent chapters.

Table 5.10: Deviation (υ_h) from the reference makespan (data set of type II).

N x L x m_L	TS-Z1	H5	LV	υ^A	υ^B
20 x 2 x 2	0.0067	0.0014	-0.0293	0.0260	0.0270
20 x 2 x 4	0.0912	0.0610	-0.0070	0.0700	0.0740
20 x 2 x 6	0.0696	0.0026	-0.0604	0.0850	0.0910
20 x 4 x 2	0.0556			0.0490	0.0580
20 x 4 x 4	0.1048			0.0730	0.0850
20 x 4 x 6	0.0955			0.1000	0.1030
20 x 6 x 2	0.0669			0.0640	0.0910
20 x 6 x 4	0.0910			0.1050	0.1140
20 x 6 x 6	0.0491			0.0920	0.0980
20 x 8 x 2	0.0880			0.0690	0.0910
20 x 8 x 4	0.0909			0.0850	0.0970
20 x 8 x 6	0.0974			0.0760	0.0800
50 x 2 x 2	0.0048	0.0048	-0.0073	0.0090	0.0090
50 x 2 x 4	0.0239	0.0224	-0.0127	0.0320	0.0320
50 x 2 x 6	0.0420	0.0386	-0.0242	0.0530	0.0540
50 x 4 x 2	0.0343			0.0300	0.0340
50 x 4 x 4	0.0696			0.0510	0.0620
50 x 4 x 6	0.0856			0.0690	0.0840
50 x 6 x 2	0.0614			0.0450	0.0630
50 x 6 x 4	0.0917			0.0570	0.0890
50 x 6 x 6	0.1237			0.0780	0.1070
50 x 8 x 2	0.0749			0.0370	0.0580
50 x 8 x 4	0.1096			0.0590	0.0840
50 x 8 x 6	0.1303			0.0770	0.1060
100 x 2 x 2	0.0003	0.0025	-0.0041	0.0020	0.0030
100 x 2 x 4	0.0048	0.0056	-0.0102	0.0120	0.0120
100 x 2 x 6	0.0201	0.0253	-0.0118	0.0300	0.0300
100 x 4 x 2	0.0196			0.0120	0.0130
100 x 4 x 4	0.0330			0.0290	0.0330
100 x 4 x 6	0.0272			0.0500	0.0400
100 x 6 x 2	0.0273			0.0270	0.0410
100 x 6 x 4	0.0593			0.0370	0.0540
100 x 6 x 6	0.0938			0.0430	0.0680
100 x 8 x 2	0.0443			0.0250	0.0440
100 x 8 x 4	0.0748			0.0350	0.0540
100 x 8 x 6	0.0856			0.0530	0.0730
150 x 2 x 2	0.0000	0.0006	-0.0029	0.0030	0.0030
150 x 2 x 4	0.0068	0.0096	-0.0034	0.0120	0.0120
150 x 2 x 6	0.0130	0.0195	-0.0023	0.0200	0.0200
150 x 4 x 2	0.0096			0.0080	0.0100
150 x 4 x 4	0.0268			0.0190	0.0220
150 x 4 x 6	0.0379			0.0280	0.0340
150 x 6 x 2	0.0236			0.0220	0.0330
150 x 6 x 4	0.0445			0.0290	0.0420
150 x 6 x 6	0.0596			0.0400	0.0540
150 x 8 x 2	0.0336			0.0270	0.0380
150 x 8 x 4	0.0499			0.0310	0.0490
150 x 8 x 6	0.0658			0.0380	0.0540
Average	0.0546	0.0162	-0.0146	0.0442	0.0547

Table 5.11: Deviation (η_h) from the lower bound (data set of type II).

N x L x m_L	TS-Z1	H5	LV	DK	η^A	η^B
20 x 2 x 2	0.0015	0.0068	0.0378	0.0084	0.0010	0.0010
20 x 2 x 4	0.0199	0.0537	0.1323	0.1248	0.0190	0.0150
20 x 2 x 6	0.0800	0.1600	0.2326	0.1631	0.0610	0.0540
20 x 4 x 2	0.0320			0.0941	0.0250	0.0150
20 x 4 x 4	0.0943			0.2241	0.0820	0.0670
20 x 4 x 6	0.0968			0.2121	0.0480	0.0440
20 x 6 x 2	0.0623			0.1401	0.0870	0.0640
20 x 6 x 4	0.1204			0.2329	0.1020	0.0910
20 x 6 x 6	0.0153			0.0683	0.0320	0.0260
20 x 8 x 2	0.1220			0.2306	0.1160	0.0900
20 x 8 x 4	0.0966			0.2062	0.1110	0.0970
20 x 8 x 6	0.0207			0.1314	0.0150	0.0100
50 x 2 x 2	0.0011	0.0010	0.0133	0.0059	0.0000	0.0000
50 x 2 x 4	0.0078	0.0093	0.0456	0.0329	0.0010	0.0010
50 x 2 x 6	0.0100	0.0136	0.0798	0.0548	0.0030	0.0020
50 x 4 x 2	0.0086			0.0449	0.0080	0.0030
50 x 4 x 4	0.0309			0.1108	0.0280	0.0160
50 x 4 x 6	0.0465			0.1447	0.0600	0.0440
50 x 6 x 2	0.0362			0.1050	0.0500	0.0300
50 x 6 x 4	0.0493			0.1562	0.1060	0.0680
50 x 6 x 6	0.0970			0.2522	0.1330	0.0970
50 x 8 x 2	0.0725			0.1604	0.0800	0.0560
50 x 8 x 4	0.1203			0.2584	0.1270	0.0980
50 x 8 x 6	0.1262			0.2947	0.1480	0.1120
100 x 2 x 2	0.0027	0.0005	0.0071	0.0030	0.0000	0.0000
100 x 2 x 4	0.0034	0.0026	0.0185	0.0083	0.0000	0.0000
100 x 2 x 6	0.0093	0.0039	0.0422	0.0301	0.0000	0.0000
100 x 4 x 2	0.0046			0.0250	0.0030	0.0020
100 x 4 x 4	0.0160			0.0508	0.0120	0.0080
100 x 4 x 6	0.0331			0.0620	0.0210	0.0120
100 x 6 x 2	0.0200			0.0488	0.0260	0.0110
100 x 6 x 4	0.0360			0.1019	0.0560	0.0370
100 x 6 x 6	0.0480			0.1571	0.0940	0.0660
100 x 8 x 2	0.0678			0.1174	0.0540	0.0340
100 x 8 x 4	0.0658			0.1520	0.0970	0.0760
100 x 8 x 6	0.1013			0.2046	0.1200	0.0970
150 x 2 x 2	0.0007	0.0001	0.0036	0.0007	0.0000	0.0000
150 x 2 x 4	0.0033	0.0004	0.0136	0.0102	0.0000	0.0000
150 x 2 x 6	0.0081	0.0015	0.0237	0.0214	0.0000	0.0000
150 x 4 x 2	0.0057			0.0156	0.0030	0.0010
150 x 4 x 4	0.0137			0.0420	0.0060	0.0020
150 x 4 x 6	0.0157			0.0559	0.0130	0.0070
150 x 6 x 2	0.0200			0.0448	0.0230	0.0120
150 x 6 x 4	0.0308			0.0789	0.0430	0.0280
150 x 6 x 6	0.0380			0.1038	0.0580	0.0430
150 x 8 x 2	0.0456			0.0824	0.0410	0.0280
150 x 8 x 4	0.0489			0.1040	0.0710	0.0510
150 x 8 x 6	0.0651			0.1404	0.0980	0.0790
Average	0.0432	0.0211	0.0542	0.1066	0.0475	0.0353

Table 5.12: Deviation (η_h) from the lower bound for data set of type II with non-identical number of machines at different stages.

N x L	TS-Z1	Cpu time	H5	LV	DK	n^A	n^B
20 x 2	0.0043	0.53	0.0067	0.0321	0.0121	0.0120	0.0090
20 x 4	0.0188	0.98			0.0506	0.0830	0.0660
20 x 6	0.0368	1.64			0.0794	0.1590	0.1280
20 x 8	0.0254	1.88			0.0587	0.1760	0.1510
50 x 2	0.0034	4.60	0.0032	0.0238	0.0175	0.0010	0.0000
50 x 4	0.0112	8.31			0.0296	0.0230	0.0120
50 x 6	0.0049	8.65			0.0068	0.1030	0.0580
50 x 8	0.0216	14.24			0.0290	0.1480	0.1150
100 x 2	0.0004	10.17	0.0001	0.0046	0.0005	0.0010	0.0000
100 x 4	0.0067	25.81			0.0118	0.0200	0.0080
100 x 6	0.0055	27.33			0.0050	0.0640	0.0400
100 x 8	0.0062	46.12			0.0111	0.1060	0.0830
150 x 2	0.0010	20.06	0.0003	0.0044	0.0010	0.0000	0.0000
150 x 4	0.0030	61.17			0.0055	0.0160	0.0070
150 x 6	0.0052	79.33			0.0088	0.0500	0.0300
150 x 8	0.0049	74.69			0.0046	0.0800	0.0640
Average	0.0100	24.09	0.0026	0.0162	0.0207	0.0651	0.0482

Chapter 6

Algorithms for FSPM/b Problem

In this chapter, we propose a tabu search procedure for solving the FSPM/b problem. As we defined earlier, FSPM/b refers to the flowshop with parallel machine problem with limited buffer sizes. Several aspects of this procedure are similar to those introduced in chapter 4 for the FSPM problem. In particular, we employ the same solution representation, neighborhood structure, and search strategy as those we used in the context of TS-Z1. The main difference however, is in the manner by which we construct a complete schedule for a given solution vector. Due to the presence of limited buffers, none of the procedures that we introduced in section 4.1 are applicable. Indeed, we note that designing an effective procedure to construct a complete schedule is significantly more complex in this case than it is in the case of the FSPM problem.

In section 6.1, we propose two procedures to construct a complete schedule for

a given solution vector. In section 6.2, we discuss the search procedure that we employed for the FSPM/b problem. In this section, we also propose appropriate methods to generate an initial solution and to deal with an infeasible solution.

6.1 Constructing a complete schedule

Given a solution vector S, we develop two different procedures to construct the corresponding complete schedule for the FSPM/b problem. The first procedure which we refer to as procedure $H1$ employs the first available machine (FAM) rule. In this procedure, a complete schedule is constructed directly from the solution vector S. The second procedure which we refer to as procedure $H2$ requires to predetermine the matrix representation that correspond to the solution vector S. In the following subsections, we discuss the two procedures.

6.1.1 Procedure $H1$: using FAM rule

In this procedure, for a given solution vector S, the corresponding job schedule at the first stage is obtained using the *first available machine* (FAM) rule. For each of the subsequent stages, the jobs are first sequenced in the increasing order of their completion time at the previous stage, and then assigned to the machines at the current stage according to the FAM rule. However, due to the limited buffer capacities, the starting time of a job on a machine at a given stage is not necessarily equal to either its completion time at the previous stage or the completion time

of the previous job on the machine at this stage. If at the time that a job gets completed on a machine the corresponding buffer area is full, then the job must remain on the machine until a buffer space becomes available. In this case, the next job which is scheduled for that machine must wait, i.e., the machine is blocked. The possibility of this situation adds to the degree of complexity of the problem, since it affects the availability of the buffer at the previous stage. Hence, all job completion times at the previous stages must be revised.

Therefore, in order to construct a complete schedule corresponding to a given solution S for the FSPM/b problem, we simulate the operation of the jobs throughout the machines at all stages. We start the simulation process by starting the first m_1 jobs in the sequence S on the m_1 parallel machines at the first stage. Since we know the processing time of each job, we can immediately determine the corresponding completion time of these m_1 jobs at the first stage. We sequence these completion times in increasing order of their values, and they constitute the initial *"event list", E*. Subsequently, throughout the simulation process we always execute the next *"event"* in the event list. An event always corresponds to the completion time of a job at a stage. We denote this completion time by t. As soon as an event occurs, i.e., a job j terminates at a given stage ℓ, we take an appropriate action accordingly, as follow.

Case a. If stage ℓ is the last stage for job j ($\ell = L$), we move job j to the list of completed jobs R and change the status of the corresponding machine at

stage ℓ to "*empty*"; this event triggers a set of subsequent events as described below.

Case b. If stage ℓ is not the last stage for job j and the subsequent buffer area is not full, we move job j to this buffer area and change the status of the corresponding machine at stage ℓ to "*empty*". Similar to case (a), this event also triggers a set of subsequent event as described below.

Case c. If stage ℓ is not the last stage for job j and the subsequent buffer area is full, we change the status of job j and the corresponding machine at stage ℓ to "*blocked*". This event does not trigger any subsequent actions and we move to the next event in the event list E.

In cases (a) and (b) above, we actually move a completed job from a machine, thus making the job as well as the machine available for subsequent processing, if any. This event allows the subsequent movement of job j to stage $\ell + 1$ or exit the shop as well as the movement of other jobs in the previous buffer areas or on blocked machines to subsequent stages. Following is a list of activities that we perform to accommodate these subsequent events. In order to allow all feasible movements of the jobs in previous stages, we perform these activities starting with stage $\ell + 1$ and working our way back to the first stage. At all times, we maintain the set of events (job completion times) in the event list in the increasing order of their values. Notice that in performing these activities, a machine is *available* if it is not blocked and it is not currently processing a job.

The following activity is applicable only if the current stage ℓ is not the last stage for job j, i.e., case (b) above.

Activity 1. At stage $\ell + 1$

If a machine is available at this stage, we move job j from the buffer area to this machine and "*free up*" the corresponding buffer space. The start time of job j at this stage is the current time, and we immediately determine its completion time. We add this completion time to the event list E and move to the previous stage (stage ℓ).

Following is the description of the activities that proceed either case (a) or (b).

Activity 2. At stage ℓ

Clearly, we have at least one machine available at this stage (since the completed job is moved to the subsequent buffer area or to the list of completed jobs, as the case may be). If there are any jobs waiting in the preceding buffer area, we start processing the job that has been there for the longest time on the available machine at stage ℓ and "free up" the corresponding buffer space. The completion time of this job at this stage can be immediately determined and added to the event list E.

Activity 3. At stage $\ell - 1$ through stage 1

We work our way back from stage $\ell - 1$ to stage 1 so that any buffer space that is "freed up" can be used by the blocked job at the preceding stage (if any). At each stage, we perform the following steps.

i. If a machine at the current stage is blocked, and there is an empty space in the subsequent buffer space, we move the blocked job to the buffer area and make the corresponding machine available.

ii. If a machine at the current stage is available and a job is waiting in the preceding buffer area, we start processing this job at the current stage and free up the corresponding buffer space. In case that more than one jobs are waiting in the preceding buffer area, we give higher priority to the job that has been there for the longest time. The completion time of the job at this stage can also be immediately determined and added to the event list E.

We work our way back to the first stage in this manner and change the status of the machines, jobs, and buffer spaces accordingly. At the first stage, if a machine becomes available, we start processing the next job in the solution vector S at this stage (unless the list is exhausted) and add its completion time to the event list. This completes all related activities to the current event. We are now prepared to start processing the "*next event*" in the event list. The simulation process terminates when the event list is empty.

In order to obtain a report of the complete schedule, each time we schedule a job on a machine at a stage, we record the machine that processes this job

and the processing order of this job on the machine. Thus, upon completion of the procedure we obtain the solution matrices I and X corresponding to the given solution vector S. In addition, when a job blocks a machine at stage ℓ, we calculate its duration of blocking as $D_{j\ell} = t - C_{j\ell}$, where t is the event time at which job j frees up this machine at stage ℓ and $C_{j\ell}$ is its completion time at this stage.

Throughout the simulation, we also maintain a list U_ℓ, for $\ell = 1, ..., L-1$, indicating the maximum number of buffer spaces actually utilized between stages ℓ and $\ell+1$. Hence, upon completing the procedure, we obtain the complete schedule as represented by $C_{j\ell}, I_{j\ell}, X_{j\ell}, D_{j\ell}$, and U_ℓ, for $j = 1, ..., N$ and $\ell = 1, ..., L-1$. The makespan (C_{max}) of the complete schedule is the time at which the last event occurs, i.e., $C_{max} = \max_j \{C_{jL}\}$.

Special case 1: more than one events occur at the same time

If more than one events occur at the same time, we order these events in decreasing order of the respective stages in which they occur. For instance, suppose at time t a job is completed at stage 2 and at the same time another job is completed at stage 4. We first process the event corresponding to stage 4, since the results of the activities corresponding to the event at stage 4 can have an impact on the subsequent activities at stage 2 (due to freeing up buffer zones and machine in between), but not vice versa.

Special case 2: zero buffer capacity

The simulation procedure that we discussed above applies to the problem with at least one buffer capacity between each pair of successive stages. In case when there is no buffer capacity allocated between two successive stages ℓ and $\ell+1$ (zero buffer capacity), we modify cases (b) and (c) as follow.

Case b′. Upon completion of job j at stage ℓ, if there is a free machine at stage $\ell+1$, we move job j to this machine at stage $\ell+1$ and "*free up*" the corresponding machine at stage ℓ. The start time of job j at stage $\ell+1$ is the current time and we immediately determine its completion time. We add this completion time to the event list E, and start the subsequent activities 2 and 3 as described above.

Case c′. If there is no free machine at stage $\ell+1$, we change the status of job j and the corresponding machine at stage ℓ to "*blocked*". Similar to case (c) above, this event does not trigger any subsequent activities and we move to the next event in the event list E.

Subsequent activities 2 and 3 are also modified in a similar manner to account for the zero buffer capacities. (Notice that activity 1 will never be executed in this special case). The key difference is that as we work our way from stage ℓ back to stage 1, if at any stage k we have a free machine and a zero buffer capacity at the preceding buffer area, we check if there is any blocked job at the preceding

Table 6.1: The job processing times.

		stage 1	2	3
job:	1	6	6	3
	2	3	1	4
	3	4	3	4
P =	4	3	4	5
	5	3	5	3
	6	8	4	10
	7	2	2	2

stage $k-1$. If there is, then we move this job to the free machine at stage k, calculate and add its completion time to the event list E, and free up the machine at the preceding stage. If there are more than one blocked jobs at stage $k-1$, we select the job that has been blocked for the longest time to move to stage k. Upon reaching the first stage, we take the similar action as we described for activity 3 above.

In the following example, we illustrate the implementation of procedure $H1$ with zero buffer capacities. An illustration for the case with non-zero buffer capacities is shown in appendix C.

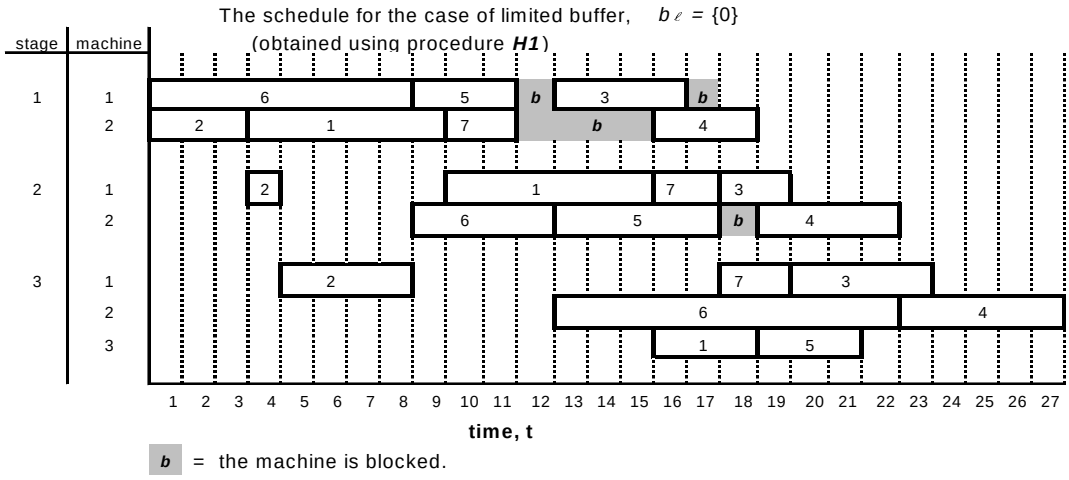
Example 2 *An illustration of the implementation of procedure* $H1$.

Consider an instance of 7 jobs with the job processing times as shown in Table 6.1. There are three stages with the number of machines $m_1 = 2$, $m_2 = 2$, and $m_3 = 3$; the buffer capacity between stages is zero, i.e., $b_1 = b_2 = 0$. Given a solution vector $S = \{6, 2, 1, 5, 7, 3, 4\}$ we intend to construct a complete schedule using procedure $H1$. The step-by-step of the procedure is shown in Table 6.2 and

Table 6.2: The step-by-step of the implementation of procedure $H1$.

Time	Jobs processed on a machine *)							Event list	Event	Completed jobs
	ℓ = 1		ℓ = 2		ℓ = 3					
t	m_1	m_2	m_1	m_2	m_1	m_2	m_3	*E*	time	*R*
0	6							$\{C_{61}\}$	{8}	
0	6	2						$\{C_{21},C_{61}\}$	{3,8}	
3	6		2					$\{C_{22},C_{61}\}$	{4,8}	
	6	1	2					$\{C_{22},C_{61},C_{11}\}$	{4,8,9}	
4	6	1			2			$\{C_{23},C_{61},C_{11}\}$	{8,8,9}	
8	6	1						$\{C_{61},C_{11}\}$	{8,9}	{2}
		1		6				$\{C_{11},C_{62}\}$	{9,12}	{2}
	5	1		6				$\{C_{11},C_{51},C_{62}\}$	{9,11,12}	{2}
9	5		1	6				$\{C_{51},C_{62},C_{12}\}$	{11,12,15}	{2}
	5	7	1	6				$\{C_{51},C_{71},C_{62},C_{12}\}$	{11,11,12,15}	{2}
11	5**b**	7	1	6				$\{C_{71},C_{62},C_{12}\}$	{11,12,15}	{2}
	5**b**	7**b**	1	6				$\{C_{62},C_{12}\}$	{12,15}	{2}
12	5**b**	7**b**	1			6		$\{C_{12},C_{63}\}$	{15,22}	{2}
		7**b**	1	5		6		$\{C_{12},C_{52},C_{63}\}$	{15,17,22}	{2}
	3	7**b**	1	5		6		$\{C_{12},C_{31},C_{52},C_{63}\}$	{15,16,17,22}	{2}
15	3	7**b**		5		6	1	$\{C_{31},C_{52},C_{13},C_{63}\}$	{16,17,18,22}	{2}
	3		7	5		6	1	$\{C_{31},C_{72},C_{52},C_{13},C_{63}\}$	{16,17,17,18,22}	{2}
	3	4	7	5		6	1	$\{C_{31},C_{72},C_{52},C_{13},C_{41},C_{63}\}$	{16,17,17,18,18,22}	{2}
16	3**b**	4	7	5		6	1	$\{C_{72},C_{52},C_{13},C_{41},C_{63}\}$	{17,17,18,18,22}	{2}
17	3**b**	4		5	7	6	1	$\{C_{52},C_{13},C_{41},C_{73},C_{63}\}$	{17,18,18,19,22}	{2}
	3**b**	4		5**b**	7	6	1	$\{C_{13},C_{41},C_{73},C_{63}\}$	{18,18,19,22}	{2}
		4	3	5**b**	7	6	1	$\{C_{13},C_{41},C_{73},C_{32},C_{63}\}$	{18,18,19,19,22}	{2}
18		4	3	5**b**	7	6		$\{C_{41},C_{73},C_{32},C_{63}\}$	{18,19,19,22}	{2,1}
		4	3		7	6	5	$\{C_{41},C_{73},C_{32},C_{53},C_{63}\}$	{18,19,19,21,22}	{2,1}
			3	4	7	6	5	$\{C_{73},C_{32},C_{53},C_{63},C_{42}\}$	{19,19,21,22,22}	{2,1}
19			3	4		6	5	$\{C_{32},C_{53},C_{63},C_{42}\}$	{19,21,22,22}	{2,1,7}
				4	3	6	5	$\{C_{53},C_{63},C_{42},C_{33}\}$	{21,22,22,23}	{2,1,7}
21				4	3	6		$\{C_{63},C_{42},C_{33}\}$	{22,22,23}	{2,1,7,5}
22				4	3			$\{C_{42},C_{33}\}$	{22,23}	{2,1,7,5,6}
					3	4		$\{C_{33},C_{43}\}$	{23,27}	{2,1,7,5,6}
23						4		$\{C_{43}\}$	{27}	{2,1,7,5,6,3}
27								{-}	{-}	{2,1,7,5,6,3,4}

*) = index **b** following a job number means that the job is blocking the machine.



Vector representation : ***S =*** {6,2,1,5,7,3,4}

Matrix representation:

I =

job:	stage 1	2	3
1	2	1	3
2	2	1	1
3	1	1	1
4	2	2	2
5	1	2	3
6	1	2	2
7	2	1	1

X =

job:	stage 1	2	3
1	2	2	1
2	1	1	1
3	3	4	3
4	4	3	2
5	2	2	2
6	1	1	1
7	3	3	2

Figure 6.1: The complete schedule generated using procedure $H1$.

the complete schedule is shown in Figure 6.1. In this figure, we also present the corresponding matrix representation.□

6.1.2 Procedure $H2$: a generic procedure

Unlike procedure $H1$ that construct a complete schedule directly from a solution vector S, procedure $H2$ requires to predetermine the matrix representation (I, X). Then, the next step is to proceed with the simulation process to construct the complete schedule. This process is similar with the process described for procedure $H1$. However, we should notice that the matrix representation (I, X) has been predetermined. As such, we do not follow the FAM rule for allocating the jobs to the machines. Instead, we use the allocation and order as prescribed by the matrices I and X, respectively.

For example, suppose at a given time t, we are about to schedule a job a at stage ℓ. At this time t, we determine if the "*designated machine*" for job a at stage ℓ, i.e., machine $I_{a\ell}$, is available. In addition, we also determine whether processing job a on machine $I_{a\ell}$ at time t does not violate the processing order that have been predetermined (as defined by the corresponding entry in the matrix X). In other words, we determine whether all jobs which are designated to be processed before job a on machine $I_{a\ell}$, i.e., all jobs $j = \{j \in (1, ..., N) : X_{j\ell} < X_{a\ell}$ and $I_{j\ell} = I_{a\ell}\}$, have been completed. If this condition is satisfied, then we schedule job a at stage ℓ; otherwise, we either move job a to the following buffer area or let it block the

machine, as the case may be.

We note that the solution matrices I and X obtained might not be feasible for the FSPM/b problem. Thus, upon completing the simulation, we investigate whether or not the given solution is feasible.

In the following discussion, we describe the simulation process which is based on the predetermined I and X matrices. Subsequently, we present the method to generate the solution matrices I and X and the method to determine the feasibility of the given solution.

We start the simulation process by starting all jobs designated to be processed first on the m_1 machines at the first stage, i.e., all jobs $j = \{j \in (1, ..., N) : X_{j1} = 1\}$. Since the processing time of each job is known, we can obtain the corresponding completion time of the jobs processed first on the machine at the first stage. Then, we sequence these completion times in increasing order of their values, and they constitute the *"event list", E*. Subsequently, throughout the simulation process we execute the next *"event"* in the event list. As soon as an event occurs, i.e., a job j terminates at a given stage ℓ, we take the appropriate action accordingly.

Case a. If stage ℓ is the last stage for job j ($\ell = L$), we move job j to the list of completed jobs R, and change the status of the corresponding machine at stage ℓ to "*empty*".

Case b. If stage ℓ is not the last stage for job j and the subsequent buffer area is not full, we move job j to this buffer area and change the status of the

corresponding machine at stage ℓ to "*empty*". However, if job j is the last job to be processed on the machine, we do not move job j to the buffer area, even though there is an empty buffer space in the buffer area. Job j will simply block the machine. (By this manner, we treat the machine as a buffer space. Thus, this allows the available buffer space to be used by other jobs). We change the status of job j and the corresponding machine at stage ℓ to "*blocked*".

Case c. If stage ℓ is not the last stage for job j and the subsequent buffer area is full, we change the status of job j and the corresponding machine at stage ℓ to "*blocked*".

Each of the three cases above triggers a set of subsequent activities as described below. In order to allow all feasible movement of the jobs at the previous stages, we perform these activities starting with stage $\ell + 1$ and working our way back to the first stage. At all times, we maintain the set of events (job completion times) in the event list in the increasing order of their values. In performing these activities, a machine is *available* if it is not blocked and it is not currently processing a job.

The following activity is applicable only if the current stage ℓ is not the last stage for job j, i.e., cases (b) and (c) above.

Activity 1. At stage $\ell + 1$

If at this stage the machine designated for job j (say machine i, where $i = I_{j,\ell+1}$)

is available and all jobs to be processed before job j on this machine have been completed, we move job j to machine i. Accordingly, we change the status of the corresponding buffer space or machine (where job j resides before moving) at stage ℓ to "*empty*". The start time of job j at stage $\ell + 1$ is the current time, and we immediately determine its completion time. We add this completion time to the event list E.

The following activities apply to all of the three cases above, i.e., cases (a), (b), and (c). (Notice that these activities also applies to case (c), since the blocked job could be the job to be processed next on a machine at the next stage following the I and X matrices). We perform the following steps only if job j is moved from the machine at the current stage ℓ (either to the designated machine at the next stage or to the subsequent buffer area).

Activity 2. At stage ℓ

Clearly, upon moving job j out of this stage, there is a machine available at this stage (say machine i, where $i = I_{j\ell}$). We inspect whether the job to be processed next on this machine (say job a as identified by the corresponding entries of the matrices I and X) has been completed at the previous stage (this job could be in the preceding buffer area or it might be blocking a machine at the preceding stage). If so, and if processing job a does not violate the processing order on machine i at stage ℓ, we move job a to machine i at this stage. Accordingly, we change the

status of the corresponding buffer space or machine (where job a resides at the previous stage) to "*empty*". The start time of job a at this stage is the current time, and we immediately determine its completion time. We add this completion time to the event list E.

Activity 3. At stage $\ell - 1$ through stage 1

We work our way back from stage $\ell - 1$ to stage 1 so that any buffer space that is freed up can be used by the blocked jobs at the preceding stages and any blocked machine which is freed can be used by a job designated to this machine. At each stage, we perform the following steps.

i. If a machine at the current stage is blocked, and there is an empty space in the subsequent buffer area, we move the blocked job to the buffer area and make the corresponding machine available. However, if the blocked job is the last job to be processed on a machine, we do not move this job to the buffer area. The blocked job will remain on the machine (with the similar reason as described in case (b)).

ii. If a machine is available at this stage (say machine i), we inspect whether the job to be processed next on this machine (say job a as identified by the corresponding entries of the matrices I and X) has been completed at the previous stage (this could be a job in the preceding buffer area or a job blocking a machine at the previous stage). If so, and if processing job

a on machine i does not violate the processing order on machine i at this stage, we move job a to machine i. Accordingly, we change the status of the corresponding buffer or machine (where job a resides at the previous stage) to "*empty*". The start time of job a at this stage is the current time, and we immediately determine its completion time. We add this completion time to the event list E.

Throughout these activities, we add an appropriate set of job completion times to the event list E. At the first stage, if a machine becomes available, we start processing the job which is scheduled to be processed next as determined by the matrices I and X, and add its completion time to the event list. This completes all activities related to the current event and we start processing the "*next event*" in the event list. Similar to procedure $H1$, if more than one events occur at the same time, we order these events in decreasing order of their respective stages. The simulation process terminates when the event list is empty.

During the simulation, we determine the duration of job j blocking a machine at stage ℓ ($D_{j\ell}$) in the similar manner as described for procedure $H1$. We also maintain a list U_ℓ, for $\ell = 1, ..., L-1$, indicating the maximum number of buffer spaces actually utilized between stages ℓ and $\ell + 1$. Therefore, upon completion of the procedure and if the solution is feasible, we obtain the complete schedule as represented by $C_{j\ell}, I_{j\ell}, X_{j\ell}, D_{j\ell}$, and U_ℓ, for $j = 1, ..., N$ and $\ell = 1, ..., L-1$. The makespan (C_{max}) of the complete schedule is the time at which the last event

occurs, i.e., $C_{max} = \max_j \{C_{jL}\}$.

Observing procedure $H2$, we can see that this procedure is a generic procedure to construct a complete schedule for the FSPM/b problem. Provided the machine allocation matrix (I) and the processing sequence matrix (X) of each job at each stage are given, we can implement procedure $H2$ to construct the complete schedule for the FSPM/b problem.

In the following subsection, we discuss the method that we used to construct the matrices I and X corresponding to a given solution S. We also explain how to determine whether or not a given matrix representation (I, X) is feasible for the corresponding FSPM/b problem.

Constructing the I and X matrices for a given solution S

Various procedures can be designed to construct a matrix representation (I, X) corresponding to a solution vector S in the context of the FSPM/b problem. Clearly, procedures $Z1$ and $Z2$ that we discussed earlier for the FSPM problem can also be employed here.

In this research, we employ procedure $Z1$, since it is the most effective procedure in the context of the tabu search for the FSPM problem. Using this procedure, for a given solution vector S, we construct a complete schedule corresponding to S assuming there are unlimited buffer sizes between stages. As a result, we obtain the following matrices: the job completion times matrix (C), the machine allocation

matrix (I) and the processing order matrix (X). Since the job completion time matrix C obtained in this manner corresponds to the unlimited buffer capacities, it might not be realized in the case of limited buffer capacities. As such, we do not utilize this matrix. On the other hand, we employ the resulting machine allocation matrix (I) and the processing order matrix (X) to construct a complete schedule for the FSPM/b problem. But, the resulting matrix representation might not be feasible for the corresponding FSPM/b problem, due to the presence of limited buffer capacities. (If the matrix is to guarantee to yield a feasible solution, the guarantee must be from the manner in which I and X matrices are generated).

One possibility to generate a solution is to generate a solution matrices I and X such that the flow of the jobs on the machines throughout the stages follows the FAM rule. The resulting schedule will always be feasible. However, this principle is essentially the same as the one that we employed in procedure $H1$. Thus, the challenge in the FSPM/b problem is how to generate a matrix representation corresponding to a solution vector S such that it is feasible. This is part of the future research, i.e., to device a procedure (other than FAM procedure) which guarantees that the solution obtained is feasible for the FSPM/b problem.

Feasibility of a given matrix representation

Given a schedule which is represented by the I and X matrices (corresponding to a solution vector S), we refer to this schedule as a "feasible" schedule if upon

termination of the simulation procedure described above, all of the jobs are processed through all stages (following the machine allocation and processing order represented by the I and X matrices). Accordingly, a solution is determined to be "infeasible" if upon termination of the simulation procedure some jobs are not completed, as discussed in the following proposition. (Recall that R is the list of completed jobs, i.e., a list that contains the jobs that have been processed through all stages.)

Proposition 2 *If upon termination of procedure H2 we determine that $|R| = N$, then the schedule which is identified by the given matrices I and X (corresponding to the solution vector S) must be feasible. Otherwise, if upon termination of procedure H2 we have $|R| < N$, then the schedule identified by I and X matrices must be infeasible.*

To show that the proposition is valid, consider the following argument.

As described in cases (b) and (c) above, when a job j is completed at stage ℓ, for $\ell < L$, we move job j to a buffer or job j blocks machine $I_{j\ell}$, if the subsequent buffer area is full. Then, following activity 1 (at stage $\ell + 1$), if we are "able"[1] to move job j to the designated machine at the next stage, i.e., machine $I_{j,\ell+1}$, we add its completion time at the next stage $C_{j,\ell+1}$ to the event list E. Eventually we execute an event that corresponds to $C_{j,\ell+1}$, and this job j keeps moving forward

[1]By "able" we mean that the machine is not blocked and it is not currently processing a job; in addition, the processing order follows matrix X.

within the flowshop.

However, if at this time we are not "able" to move job j to machine $I_{j,\ell+1}$ (due to the fact that some other jobs that must be processed before job j on the same machine at stage $\ell+1$ are not processed yet), then job j will either remain in the buffer space or it will remain blocking the machine at stage ℓ, as the case may be. As the simulation progresses, at a given event we may execute activity 2 and activity 3. At this time, either one of following two cases will occur.

Case 1. We are "able" to move job j to the designated machine at stage $\ell+1$.

In this case, upon moving job j to the designated machine at stage $\ell+1$, we add its completion time at stage $\ell+1$, i.e., $C_{j,\ell+1}$, to the event list E, and this job continues its normal progression through the flowshop. If this situation occurs to all jobs that cannot be processed immediately at the next stage upon completion at the current stage, these jobs will eventually reach the last stage. By the time the event list E is exhausted, all jobs must be completed at the last stage. Consequently, all completed jobs at the last stage must have been moved to the list of completed jobs R. Thus, we must have the completion time of each job at each stage and we must have $|R| = N$, upon termination of the procedure.

Case 2. We are still not "able" to move job j to the designated machine at stage $\ell+1$.

In this case, we do not take action on job j (except that if job j is currently blocking a machine and a buffer space becomes available at the subsequent buffer

area, we move job j to this buffer space). If at a subsequent event we are finally "able" to move job j to the machine at stage $\ell+1$, case 1 above will apply to job j. However, if the simulation procedure terminates and we remain "unable" to move job j to the designated machine at the next stage, the following conditions must be true.

- There must be a job k that has to be processed before job j on the same machine at stage $\ell+1$, i.e., $I_{k,\ell+1} = I_{j,\ell+1}$ and $X_{k,\ell+1} < X_{j,\ell+1}$. (If there is no such job k, job j would have been started at stage $\ell+1$ as soon as the corresponding machine at stage $\ell+1$ were available after its completion at stage ℓ).

- The machine designated for job k at stage ℓ, i.e., machine $I_{k\ell}$, must be blocked by a job. Therefore, job k has never been started at stage ℓ. (Had this machine not been blocked, job k could have been started and completed at stage ℓ. Eventually, it would have started and completed at stage $\ell+1$. This would have allowed job j to be started and completed at stage $\ell+1$).

From the above conditions, we make the following observations.

- Since job k has never been started at stage ℓ, all jobs following jobs k on the same machine can not be started either. Consequently, there will be no event corresponding to the completion time of job k and all jobs following job k on the same machine at stage ℓ. Following the flowshop rule, there will be no

event corresponding to the completion time of these jobs at the subsequent stages.

- Similarly, since we are never "able" to move job j to the designated machine at stage $\ell + 1$, there will be no event corresponding to the completion time of job j at stage $\ell + 1$. Consequently, there will be no event corresponding to the completion time of all jobs following job j on the same machine at stage $\ell + 1$. Following the flowshop rule, there will be no event corresponding to the completion time of these jobs at the subsequent stages.

Therefore, it is obvious that all these jobs (corresponding to the two observations above) will never reach the last stage. Since R contains only jobs that have reached the last stage, by the time the event list E is exhausted, i.e., the procedure terminates, it must be true that $|R| < N$.

To illustrate the implementation of procedure $H2$ and the method of determining an infeasible solution, we present the following example.

Example 3 *An illustration of the implementation of procedure $H2$.*

Consider the same instance of 7 jobs as shown in Table 6.1, i.e., 3-stages FSPM/b problem with the number of machines $m_1 = m_2 = 2$, $m_3 = 3$ and buffer sizes $b_1 = b_2 = 0$.

Given the similar solution vector $S = \{6, 2, 1, 5, 7, 3, 4\}$, we intend to construct a complete schedule. Using procedure $Z1$ (assuming to have unlimited buffer

capacities), we obtain the I and X matrices corresponding to the solution vector S. These matrices are shown in Figure 6.2. Then, using procedure $H2$, we construct a complete schedule with the limited buffer capacities. The step-by-step of the implementation of procedure $H2$ is shown in Table 6.3. The resulting (infeasible) schedule is also shown in Figure 6.2. A brief description of this implementation is given below.

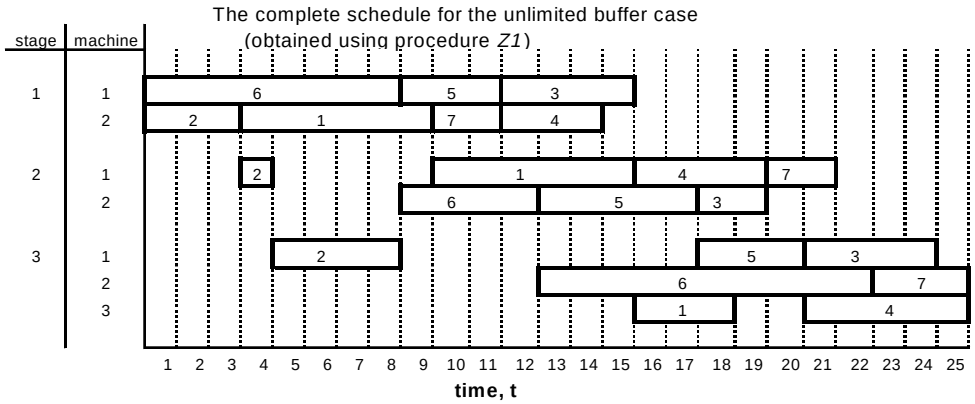
First, we schedule jobs 6 and 2 to the machines at the stage 1 and we add their completion times to the event list E, i.e., $E = \{(C_{21} = 3), (C_{61} = 8)\}$. The next event is indicated by the completion time of job 2 at time 3. We schedule job 2 to the next stage and we schedule the job following job 2 on machine 2 at stage 1, i.e., job 1. At this time, we remove C_{21} from the event list and we add C_{11} and C_{22} to the event list E. Thus, we have $E = \{(C_{22} = 4), (C_{61} = 8), (C_{11} = 9)\}$. We continue with the next event, etc.

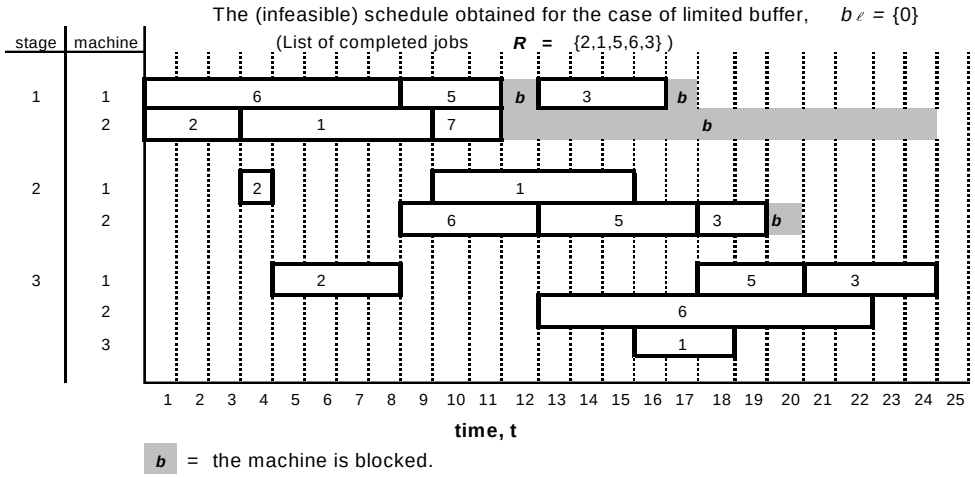
Consider the flow of job 5 within this process. At time 11, we determine that job 5 is completed at stage 1. At this time, the machine designated for job 5 at stage 2, i.e., machine 2, is not available (since it is processing job 6). Therefore, job 5 blocks the machine at stage 1. At time 12, we are "able" to move job 5 to machine 2 at stage 2 and we add its completion time to the event list. At time 17, job 5 is completed at stage 2 and we move this job to the next stage. Finally, this job is completed at the last stage at time 20 and we move it to the list of completed jobs R. Observe that the flow of job 5 throughout the simulation process illustrates

Table 6.3: The step-by-step of the implementation of procedure $H2$.

Time	Jobs processed on a machine *)							Event list	Event	Completed jobs
	ℓ = 1		ℓ = 2		ℓ = 3					
t	m_1	m_2	m_1	m_2	m_1	m_2	m_3	*E*	time	*R*
0	6							$\{C_{61}\}$	{8}	
0	6	2						$\{C_{21},C_{61}\}$	{3,8}	
3	6		2					$\{C_{22},C_{61}\}$	{4,8}	
	6	1	2					$\{C_{22},C_{61},C_{11}\}$	{4,8,9}	
4	6	1			2			$\{C_{23},C_{61},C_{11}\}$	{8,8,9}	
8	6	1						$\{C_{61},C_{11}\}$	{8,9}	{2}
		1		6				$\{C_{11},C_{62}\}$	{9,12}	{2}
	5	1		6				$\{C_{11},C_{51},C_{62}\}$	{9,11,12}	{2}
9	5		1	6				$\{C_{51},C_{62},C_{12}\}$	{11,12,15}	{2}
	5	7	1	6				$\{C_{51},C_{71},C_{62},C_{12}\}$	{11,11,12,15}	{2}
11	5**b**	7	1	6				$\{C_{71},C_{62},C_{12}\}$	{11,12,15}	{2}
	5**b**	7**b**	1	6				$\{C_{62},C_{12}\}$	{12,15}	{2}
12	5**b**	7**b**	1			6		$\{C_{12},C_{63}\}$	{15,22}	{2}
		7**b**	1	5		6		$\{C_{12},C_{52},C_{63}\}$	{15,17,22}	{2}
	3	7**b**	1	5		6		$\{C_{12},C_{31},C_{52},C_{63}\}$	{15,16,17,22}	{2}
15	3	7**b**		5		6	1	$\{C_{31},C_{52},C_{13},C_{63}\}$	{16,17,18,22}	{2}
16	3**b**	7**b**		5		6	1	$\{C_{52},C_{13},C_{63}\}$	{17,18,22}	{2}
17	3**b**	7**b**			5	6	1	$\{C_{13},C_{53},C_{63}\}$	{18,20,22}	{2}
		7**b**		3	5	6	1	$\{C_{13},C_{32},C_{53},C_{63}\}$	{18,19,20,22}	{2}
18		7**b**		3	5	6		$\{C_{32},C_{53},C_{63}\}$	{19,20,22}	{2,1}
19		7**b**		3**b**	5	6		$\{C_{53},C_{63}\}$	{20,22}	{2,1}
20		7**b**		3**b**		6		$\{C_{63}\}$	{22}	{2,1,5}
		7**b**			3	6		$\{C_{63},C_{33}\}$	{22,24}	{2,1,5}
22		7**b**			3			$\{C_{33}\}$	{24}	{2,1,5,6}
24		7**b**						{-}	{-}	{2,1,5,6,3}

*) = index **b** following a job number means that the job is blocking the machine.





Vector representation : S = {6,2,1,5,7,3,4}

Matrix representation: (obtained using procedure *Z1*)

I =

job:	stage 1	stage 2	stage 3
1	2	1	3
2	2	1	1
3	1	2	1
4	2	1	3
5	1	2	1
6	1	2	2
7	2	1	2

X =

job:	stage 1	stage 2	stage 3
1	2	2	1
2	1	1	1
3	3	3	3
4	4	3	2
5	2	2	2
6	1	1	1
7	3	4	2

Figure 6.2: The (infeasible) schedule generated using procedure $H2$ (and procedure $Z1$).

the situation that we described in case 1 above.

Now, let us observe the flow of job 7. At time 11, job 7 is completed at stage 1, but we are not "able" to move this job to the designated machine at stage 2, i.e., machine 1 (since this machine is processing job 1). Therefore, job 7 blocks machine 2 at stage 1. At time 15, the machine which is designated for job 7 at stage 2, i.e., machine 1, becomes available. But, we are still not "able" to schedule job 7 at stage 2 since job 4 has not been completed at stage 2 (as defined in matrix representation X, job 4 should be processed before job 7 at stage 2). Similarly, we are not able to process job 4 at stage 1 because the machine is blocked by job 7. The flow of job 7 illustrates the situation that we described in case 2 above.

Following our argument as in case 2 above, we determine the following.

- Job 7 can not be started at stage 2, therefore, there must be a job that has to be processed before job 7 on the same machine at stage 2. This job is determined to be job 4.

- The machine designated for job 4 at stage 1 must be blocked. Indeed, machine 2 which is designated for job 4 at stage 1 is blocked by job 7.

- Since job 4 has never been started at stage 1, there is no event corresponding to the completion time of job 4 at stage 1 and at the subsequent stages. Similarly, since job 7 can not be started at stage 2, there is no event corresponding to the completion time of job 7 at stage 2 and at the subsequent stages.

Upon the termination of procedure $H2$ at time 24 (the event list has been exhausted), we found that only 5 jobs have been completed, i.e., $R = \{2, 1, 5, 6, 3\}$, while jobs 4 and 7 remain incomplete. Therefore, we determine that the given solution vector S is not feasible.□

6.2 Tabu Search procedure

The tabu search features and strategy that we employed for solving the FSPM/b problem are similar to those that we proposed in the context of the FSPM problem and discussed in section 4.4. We also use similar parameter values. Since we have two different procedures $H1$ and $H2$ for constructing a complete schedule, this leads to two different tabu search procedures that we refer to as TS-H1 and TS-H2, respectively. As we mentioned earlier, in algorithm TS-H2 we employ procedure $Z1$ to generate the I and X matrices corresponding to a given solution vector S.

As for the initial solution, in both algorithms we employ the same procedure as we used for the FSPM problem, i.e., we randomly generate a solution vector S.

In the following subsections, we discuss how we deal with an infeasible solution that we may encounter in the context of TS-H2.

6.2.1 Dealing with an infeasible matrix representation

As we mentioned earlier, in the context of procedure $H2$, we might encounter a schedule (as represented by matrices I and X that correspond to a given solution

vector S) which is not feasible for the problem. An infeasible schedule might be encountered either at the initial solution or through the neighborhood search.

In the latter case, our strategy is to set the corresponding makespan equals to infinity. As such, the tabu search procedure will avoid selecting an infeasible neighbor for the next iteration (due to its high cost). In case that all neighbors are infeasible, we backtrack, i.e., we move to the previous solution as the starting point for the next iteration. (Although we have the same starting point as that used in the previous iteration, we expect to move to a different neighboring solution, since we choose the neighbors randomly, as discussed in section 4.4).

In the former case, i.e., when the schedule corresponding to the *initial* solution vector S is not feasible, we discard this vector and construct a new vector randomly. However, it is possible that TS-H2 remains unable to generate a feasible initial solution, even though we repeat this process for several times. Consequently, we are not able to start the search procedure. In this circumstance, we terminate the procedure without producing any feasible solution to the problem and conclude that TS-H2 has failed to solve the instance considered.

Chapter 7

Computational Experiment for FSPM/b Problem

In this chapter, we present the computational results that we performed for the FSPM/b problem using algorithms TS-H1 and TS-H2. These algorithms are coded using Fortran90 and ran on a 266 Mhz PC. In section 7.1, we describe the data sets that we use for the experiment. In section 7.2, we discuss the performance measure that we employ to evaluate the solution quality. Finally, in section 7.3, we present the experimental results.

7.1 Data sets

For each instance of FSPM problem, we define a set of parameter values which we refer to as *the FSPM parameters*. These include the number of jobs (N), the

number of stages (L), the number of machines at each stage m_ℓ, for $\ell = 1, ...L$, and the job processing time $p_{j\ell}$, for $j = 1, ..., N$ and $\ell = 1, ..., L$. In addition to these parameter values, to identify an instance of the FSPM/b problem, we need to specify the buffer capacities $b_1, b_2, ..., b_{L-1}$.

In this experiment, we construct and solve three sets of instances that we refer to as data sets of type I, type II, and type III, respectively. Within each type of data set, we construct and solve several instances of the problem with different FSPM parameters and with different buffer capacities. The processing time ($p_{j\ell}$) for the first two data sets are randomly generated, and their structure is identical to those described for data sets of type I and type II for the FSPM problem (section 5.1), respectively. For data set of type III, we adopt the parameter values from a published article by Wittrock [66].

7.1.1 Data set of type I

In this type of instances, the job processing time ($p_{j\ell}$, for $j = 1, ..., N$ and $\ell = 1, ..., L$) for each instance is constructed randomly and according to the same pattern as for the data set of type I for the FSPM problem (section 5.1.1). As mentioned earlier, we already know the optimal solution for this type of instances for the FSPM problem and we know that at the optimal solution, the buffer utilization is zero at all stages. It follows that the solution which is optimal for an instance of the FSPM problem in this class is also optimal for the corresponding instances

of the FSPM/b problem with any (non-negative) buffer capacities $b_1, b_2, ..., b_{L-1}$.

For each set of values of the FSPM parameters, we construct four instances of the FSPM/b problem, as follow.

- Instance 1: let $b_\ell = 0$, for $\ell = 1, ..., L-1$.
- Instance 2: let $b_\ell = 1$, for $\ell = 1, ..., L-1$.
- Instance 3: let $b_\ell = \frac{1}{2}U_\ell$, for $\ell = 1, ..., L-1$.
- Instance 4: let $b_\ell = U_\ell$, for $\ell = 1, ..., L-1$.

where U_ℓ is the maximum buffer utilization at buffer zone ℓ for the corresponding FSPM problem, for $\ell = 1, ..., L-1$, as obtained via the TS-Z1 procedure.

Since the optimal schedules for the four instances are the same, our primary objective in solving these instances is to determine the impact of the limited buffer capacities on the search procedure itself.

For this type of data set, we generate instances of different sizes with the number of jobs $N = 20, 30, 40, 50$, and the number of stages $L = 2, 3, 4$. The number of machines (m_ℓ) at each stage is assumed to be identical with $m_\ell = 2, 4, 6$, for $\ell = 1, ..., L$. For each problem size within each type of data set, we construct and solve 5 instances of the problem, and present the average result obtained for the 5 instances.

7.1.2 Data set of type II

In this group of instances, we also determine the values of the job processing time ($p_{j\ell}$) randomly and according to the same pattern as we used for the corresponding data set of type II for the FSPM problem (section 5.1.2). Corresponding to each set of the FSPM parameters, we construct four different instances of the FSPM/b in the similar manner as those created for data set of type I, i.e., we set $b_\ell = 0, b_\ell = 1, b_\ell = \frac{1}{2}U_\ell$, and $b_\ell = U_\ell$, for $\ell = 1, ..., L-1$, where U_ℓ is defined similarly as we defined U_ℓ in data set of type I above.

Again, our primary objective in constructing these four instances with the same FSPM parameter values is to determine the impact of the limited buffer sizes on the performance of the search procedure. However, in the first three instances, it is conceivable that the optimal makespan for the FSPM/b problem is indeed larger than the optimal makespan for the corresponding FSPM problem. Therefore, in these instances, if the heuristic procedure finds a longer makespan, it could be either due to a correspondingly larger optimal makespan and/or due to the impact of the limited buffer capacities on the heuristic search itself.

For this type of data set, we generate instances of different sizes similar to the sizes that we stated for data set of type I, i.e., with $N = 20, 30, 40, 50$, $L = 2, 3, 4$, and $m_\ell = 2, 4, 6$, for $\ell = 1, ..., L$. For each problem size within each type of data set, we also construct and solve 5 instances of the problem, and present the average result obtained for the 5 instances.

Furthermore, for this type of data set, we also generate a set of instances where the number of machines at different stages are not necessarily identical. We generate instances of different sizes with the number of jobs $N = 20, 30, 40, 50$ and the number of stages $L = 2, 3, 4, 6, 8$. The number of machines at each stage is generated randomly following a uniform distribution between 1 and 5, i.e., $P(m_\ell = k) = \frac{1}{5}$, for $k = 1, ..., 5$, and for all ℓ. For each problem size, we construct and solve 10 instances and present the average result of the 10 instances.

Thus, together with the variation on the buffer capacities, we generate 720 different instances for data set of type I and 920 different instances for data set of type II.

7.1.3 Data set of type III

As we mentioned earlier, we adopt this data set from the open literature for the FSPM/b problem, i.e., the data set due to Wittrock [66]. In the remaining parts of this study, we also refer to this data set as the Wittrock test instances. There are a total of six instances in this data set. Each instance has $L = 3$ stages with the number of machines at each stage $m_1 = 2$, $m_2 = 3$, and $m_3 = 3$. There are *13 job types* (denoted as job types A through M), and the processing time of each job type at each stage (i.e., $p_{j\ell}$) is given in Table 7.1. Each *instance* of the problem consists of a collection of several job types. The number of each job type for each of the six instances of the problem is given in Table 7.1.

Table 7.1: Test problem data for Wittrock problems [66].

	Processing time $p_{j\ell}$ (min.)			Production requirement of each instance for each job type					
	Stage number			Problem instance					
Job type	ℓ = 1	ℓ = 2	ℓ = 3	1	2	3	4	5	6
A	39	11	14	12	-	-	-	-	-
B	13	28	54	1	-	-	-	-	-
C	22	56	60	26	-	-	14	23	20
D	234	39	0	-	-	-	2	-	-
E	39	25	80	-	6	7	4	-	1
F	13	70	54	-	14	20	16	-	-
G	143	66	0	1	4	-	-	3	1
H	0	28	14	7	-	-	-	-	-
I	26	39	74	-	6	-	-	-	5
J	18	59	34	-	4	-	-	-	-
K	22	70	40	4	-	-	-	-	-
L	13	70	54	-	4	-	-	-	-
M	61	46	34	-	-	11	-	14	3
			total:	51	38	38	36	40	30

As a part of the problem, Wittrock also considers the transport time to move a job from one stage to the next, which is defined as one minute. Since the transport time between stages is not included in our procedure to construct a complete schedule, we incorporate the transport time by adding it to the processing time of the jobs at stages 2 and 3. Thus, in the instances that we solve, we set the processing time of job j at stage ℓ as $p'_{j\ell} = p_{j\ell} + 1$, for $j = 1, ..., N$ and $\ell = 2, 3$. (Notice that there are some jobs that do not require processing at stage 3, i.e., $p_{j3} = 0$. For these jobs, we assume that they leave the production line as soon as they are completed at stage 2. As such, at stage 3 we add the transport time of one unit only to the jobs which require processing at that stage, i.e., jobs with $p_{j3} > 0$). By adding the transport time to the job processing time, the problem

changes slightly and it becomes more restricted, as described below.

Consider, for example, a job j which is just completed at time t on machine i at stage 2. In the original problem, the completion time of job j at stage 2 is equal to t, i.e., $C_{j2} = t$. If there is a job ready to be processed at stage 2 (say job k), then we can start processing job k on machine i at time t and its completion time at stage 2 is $C_{k2} = t + p_{k2}$. In our approach, we incorporate the travel time (1 minute) of a job between stages. Therefore, when job j is completed at time t on machine i at stage 2, its completion time becomes $C_{j2} = t + 1$. Consequently, we are not able to start processing job k at time t on machine i at stage 2. Instead, we start job k at time $t + 1$ on machine i at stage 2 and its completion time at stage 2 is $C_{k2} = t + 1 + p_{k2}$, which is one unit larger than its original completion time.

In performing the experiment, Wittrock considers two cases. The first case assumes unlimited buffer capacity in front of each machine. In the second case, Wittrock considers only a buffer capacity of one in front of each machine.

In our experiment, we follow the scenario used by Wittrock. First, we consider the case with unlimited buffer capacities, then we consider the limited buffer case. For the case with limited buffer capacities, we define $b_\ell = \{3, 3\}$. In addition to the instances which are solved by Wittrock, we also perform an experiment with $b_\ell = \{0, 0\}$.

7.2 Performance measure

In order to determine the quality of the solution obtained, we compare the makespan obtained using the proposed algorithms with some reference values. For data set of type I, we know the optimal solution to the problem. Therefore, we measure the solution quality by evaluating the deviation of the makespan obtained from the corresponding optimal value. The deviation is measured in a manner which is similar to the case of the FSPM problem as discussed in section 5.4.1.

$$\delta_h = \frac{C_h - C_{opt}}{C_{opt}}$$

where

δ_h = The performance ratio of algorithm h

C_h = The makespan obtained using the proposed algorithm h

C_{opt} = The optimal makespan

For the case of the data set of type II as well as the Wittrock test instances, their corresponding optimal makespan are not known. Therefore, to evaluate the experimental results, we compare the makespan obtained with the lower bound of the corresponding optimal solution. The evaluation is based on the deviation of the makespan obtained using the proposed algorithms from the best lower bound, and is computed as follow.

$$\eta_h = \frac{C_h - LB}{LB}$$

where

η_h = The performance ratio of algorithm h

C_h = The makespan obtained using proposed algorithm h

LB = The best lower bound

As for the lower bounds, it is true that the lower bounds that we discussed for the FSPM problem are also valid for the FSPM/b problem, although they may be further away from the corresponding optimal values. In our experiment with data set of type II, we employ these lower bounds in the context of FSPM/b problem, and discuss their relative strength.

For the data set of type III, in his experiment Wittrock uses the maximum machine workload (w^*) as the lower bound to the optimal solution, which is defined to be the maximum of the average workload among machines at each stage, i.e., $w^* = \max_{\ell} \{\sum_j \frac{p_{j\ell}}{m_\ell}\}$. Observe that w^* is equivalent to T_ℓ in the formulation of LB_2 (section 5.2). We can easily verify that w^* is always dominated by the lower bound that we developed for the FSPM problem as discussed in section 5.2. Therefore, in our experiment with the data set of type III (i.e., Wittrock test instances), we also employ the lower bounds as discussed in section 5.2 to evaluate the solution quality.

7.3 Experimental results

In this experiment, our objective is to study the effectiveness of the proposed algorithms TS-H1 and TS-H2 in solving the FSPM/b problem. In performing the experiment, we run the program until the stopping criterion (as discussed in section 4.4) is met and retain the best makespan obtained. We construct the initial solution vector for each of the two procedures randomly. For TS-H2, if the initial random vector results in an infeasible solution (see section 6.1.2), we discard this solution and construct another solution vector randomly. We continue this process until a feasible solution vector is obtained. If after 10,000 attempts we fail to generate a feasible solution vector, we terminate the procedure and we declare that procedure $H2$ fails to solve that instance of the problem.

In the following subsections, we present the experimental results for each type of data set separately.

7.3.1 Data set of type I

As we described earlier, the optimal solution and the optimal buffer configuration of the instances in this category are known. In this experiment, we run the program with $b_\ell = 0, b_\ell = 1, b_\ell = \frac{1}{2}U_\ell$, and $b_\ell = U_\ell$, for all $\ell = 1, ..., L-1$.

The experimental results are shown in Tables 7.2 to 7.4. In each table, the values shown represent the average values for the 5 instances. The first column in each table shows the size of the instances, i.e., the number of jobs, the number of

stages, and the number of machines at each stage.

In Table 7.2, we present the makespans obtained using algorithms TS-H1 and TS-H2 for various buffer allocations. As for comparison, in this table we also present the results obtained assuming unlimited buffer sizes between stages, i.e., FSPM problem (obtained using TS-Z1). The performance ratio and the CPU time corresponding to this table are shown in Tables 7.3 and 7.4, respectively. Notice that in these tables, in some instances under TS-H2, we do not present the results. This implies the fact that in one or more of the corresponding instances, TS-H2 fails to generate a feasible solution.

From these tables, we make the following observations.

1. From the results obtained using TS-H1, we can see that the average performance ratio for $b_\ell = 0, 1, \frac{1}{2}U_\ell$, and U_ℓ, are 0.0621, 0.0429, 0.0423, and 0.0380, respectively. The average performance ratio is getting better (smaller) as we increase the buffer capacity. Since we know that the optimal makespan of the instances with different buffer capacities is the same, this indicates that the performance of TS-H1 diminishes as we limit the buffer sizes. However, this phenomenon is not consistent among all instances, since in some instances, we observe that the resulting makespan actually decreases as we reduce the buffer capacities. Apparently, in these instances the limitation on the buffer sizes is actually helpful to the search procedure.

2. If we compare the results obtained via TS-Z1 for the unlimited buffer sizes

with the results of TS-H1 with $b_\ell = U_\ell$, we observe that in 10 out of 36 instances TS-H1 obtains smaller makespan. There can be two different explanations for this phenomenon. The first explanation is that in these instances, the limitation on the buffer sizes is actually helpful to the search procedure by making some of the neighborhood solutions unattractive for the tabu search (as in the previous observation). The second explanation is that indeed the FAM rule used in TS-H1 is superior to the $Z1$ procedure of TS-Z1. As mentioned in section 4.1.2, a preliminary study showed that procedure $Z1$ is generally better than the FAM rule in the context of the FSPM problem with identical number of machines at all stages. Thus, it follows that this phenomenon is likely to be due to the limitation on the buffer sizes.

3. Similar to our observation for FSPM problem, the performance ratio tends to increase as the number of stages increase and as the number of jobs increase. The effect of the number of machines at each stage (m_ℓ) on the performance ratio does not seem to follow a particular pattern.

4. The number of instances in which TS-H2 fails to generate a feasible solution increases as the buffer capacities decrease. The highest number of failures occur when $b_\ell = 0$, for all ℓ.

5. Compared with TS-Z1, both TS-H1 and TS-H2 have larger CPU time, as expected.

Table 7.2: The makespan obtained using TS-H1 and TS-H2 (data set of type I).

	$b_\ell = 0$		$b_\ell = 1$		$b_\ell = ½\ U_\ell$		$b_\ell = U_\ell$		$b_\ell = \infty$
N x L x m_L	TS-H1	TS-H2	TS-H1	TS-H2	TS-H1	TS-H2	TS-H1	TS-H2	TS-Z1
20 x 2 x 2	1199.60	1199.80	1183.60	1184.40	1186.20	1188.60	1183.20	1185.00	1184.20
20 x 2 x 4	509.40	510.80	511.00	508.00	509.00	506.40	511.40	504.80	505.60
20 x 2 x 6	354.00	347.40	353.20	350.00	351.80	350.00	353.00	350.80	350.60
20 x 3 x 2	1238.80	1242.40	1211.40	1207.20	1225.20	1221.80	1212.20	1220.00	1211.00
20 x 3 x 4	542.60	537.60	536.80	536.20	539.60	530.80	537.00	530.20	533.00
20 x 3 x 6	420.80	416.00	419.80	412.00	421.40	416.40	422.80	417.00	417.40
20 x 4 x 2	1353.80	1330.00	1296.00	1302.60	1325.00	1325.00	1296.80	1305.20	1294.00
20 x 4 x 4	554.40	558.00	555.00	559.20	556.00	562.00	552.60	554.40	555.00
20 x 4 x 6	432.40	429.80	433.40	431.60	428.00	431.00	429.00	430.40	429.20
30 x 2 x 2	2052.00	-	2013.80	2013.00	2009.60	2013.20	2009.60	2007.80	2004.00
30 x 2 x 4	860.60	864.00	854.40	854.80	852.00	848.80	849.80	847.40	853.00
30 x 2 x 6	515.60	516.60	513.40	507.40	510.20	507.20	512.00	507.20	509.80
30 x 3 x 2	2230.60	-	2170.40	2168.60	2163.20	2162.60	2165.60	2154.00	2157.20
30 x 3 x 4	864.00	875.00	856.20	857.20	859.20	852.20	856.80	848.20	854.40
30 x 3 x 6	563.00	556.80	557.00	549.20	554.00	549.20	552.20	555.60	552.00
30 x 4 x 2	2452.80	-	2345.20	-	2330.20	2328.20	2317.20	2328.20	2334.80
30 x 4 x 4	908.40	902.20	883.40	882.40	893.00	890.00	893.20	879.20	881.80
30 x 4 x 6	662.00	662.40	662.60	653.80	657.80	655.40	657.20	652.40	646.00
40 x 2 x 2	3472.80	-	3395.60	-	3396.00	-	3383.40	3386.60	3382.40
40 x 2 x 4	1200.00	-	1179.40	1181.20	1180.40	1174.40	1173.40	1178.20	1177.60
40 x 2 x 6	697.20	695.00	691.40	684.40	692.00	683.80	688.80	685.40	687.00
40 x 3 x 2	3615.80	-	3466.80	-	3449.40	-	3442.60	3445.60	3426.40
40 x 3 x 4	1231.20	-	1208.40	1208.20	1206.80	1200.00	1197.40	1197.40	1194.40
40 x 3 x 6	783.40	791.60	781.20	772.80	783.80	769.40	778.20	775.60	768.40
40 x 4 x 2	3610.00	-	3489.60	-	3478.60	-	3449.00	3428.80	3406.80
40 x 4 x 4	1317.80	-	1284.80	1284.40	1291.00	1276.60	1274.00	1267.00	1269.80
40 x 4 x 6	818.00	823.40	808.20	808.40	812.20	804.60	808.60	805.40	802.20
50 x 2 x 2	4888.40	-	4743.40	-	4741.20	-	4707.60	4708.60	4721.00
50 x 2 x 4	1685.60	-	1663.20	-	1652.60	1649.20	1648.20	1645.00	1654.80
50 x 2 x 6	959.80	980.80	949.20	946.60	951.20	943.60	947.40	945.00	944.20
50 x 3 x 2	4969.00	-	4743.60	-	4778.80	-	4729.00	4702.40	4694.80
50 x 3 x 4	1771.20	-	1730.60	-	1708.60	1723.40	1703.40	1703.60	1699.00
50 x 3 x 6	995.40	-	983.60	972.60	974.40	965.40	974.20	970.40	982.60
50 x 4 x 2	5521.80	-	5174.00	-	5208.80	-	5130.60	-	5088.20
50 x 4 x 4	1878.40	-	1843.60	-	1831.20	-	1820.60	1811.80	1801.80
50 x 4 x 6	1017.40	-	1001.80	988.00	989.00	981.60	988.40	991.60	989.00
Average *)	1615.22	-	1569.31	-	1569.37	-	1559.90	-	1554.54
Average **)	749.36	749.45	919.21	916.32	1021.15	1017.61	1457.88	1455.03	

*) = based on the average of all instances.

**) = based on the average of instances where TS-H2 gives a feasible solution.

Table 7.3: The performance ratio obtained using TS-H1 and TS-H2 (data set of type I).

	$b_\ell = 0$		$b_\ell = 1$		$b_\ell = ½\, U_\ell$		$b_\ell = U_\ell$		$b_\ell = \infty$
N x L x m_L	TS-H1	TS-H2	TS-H1	TS-H2	TS-H1	TS-H2	TS-H1	TS-H2	TS-Z1
20 x 2 x 2	0.0293	0.0290	0.0153	0.0160	0.0172	0.0192	0.0149	0.0163	0.0160
20 x 2 x 4	0.0317	0.0341	0.0348	0.0286	0.0309	0.0254	0.0355	0.0221	0.0239
20 x 2 x 6	0.0427	0.0244	0.0403	0.0315	0.0376	0.0310	0.0405	0.0340	0.0341
20 x 3 x 2	0.0486	0.0512	0.0252	0.0219	0.0365	0.0332	0.0258	0.0327	0.0251
20 x 3 x 4	0.0571	0.0483	0.0469	0.0444	0.0522	0.0348	0.0473	0.0342	0.0388
20 x 3 x 6	0.0683	0.0555	0.0658	0.0470	0.0701	0.0566	0.0732	0.0591	0.0590
20 x 4 x 2	0.0734	0.0552	0.0285	0.0331	0.0508	0.0512	0.0288	0.0356	0.0266
20 x 4 x 4	0.0442	0.0513	0.0455	0.0534	0.0475	0.0589	0.0410	0.0445	0.0454
20 x 4 x 6	0.0744	0.0684	0.0773	0.0727	0.0643	0.0710	0.0652	0.0690	0.0663
30 x 2 x 2	0.0376	-	0.0184	0.0179	0.0163	0.0182	0.0163	0.0153	0.0135
30 x 2 x 4	0.0400	0.0440	0.0323	0.0324	0.0291	0.0252	0.0266	0.0236	0.0299
30 x 2 x 6	0.0456	0.0475	0.0412	0.0289	0.0345	0.0288	0.0380	0.0283	0.0335
30 x 3 x 2	0.0601	-	0.0316	0.0306	0.0281	0.0280	0.0292	0.0239	0.0256
30 x 3 x 4	0.0539	0.0673	0.0445	0.0456	0.0481	0.0396	0.0453	0.0347	0.0422
30 x 3 x 6	0.0665	0.0547	0.0548	0.0403	0.0494	0.0402	0.0467	0.0522	0.0458
30 x 4 x 2	0.0865	-	0.0379	-	0.0319	0.0307	0.0257	0.0306	0.0336
30 x 4 x 4	0.0766	0.0695	0.0467	0.0452	0.0582	0.0547	0.0586	0.0417	0.0441
30 x 4 x 6	0.0681	0.0686	0.0690	0.0546	0.0610	0.0574	0.0600	0.0525	0.0423
40 x 2 x 2	0.0392	-	0.0159	-	0.0160	-	0.0121	0.0131	0.0116
40 x 2 x 4	0.0468	-	0.0289	0.0307	0.0297	0.0246	0.0235	0.0281	0.0274
40 x 2 x 6	0.0488	0.0442	0.0399	0.0291	0.0403	0.0284	0.0356	0.0302	0.0333
40 x 3 x 2	0.0762	-	0.0320	-	0.0266	-	0.0244	0.0257	0.0198
40 x 3 x 4	0.0674	-	0.0470	0.0471	0.0457	0.0398	0.0379	0.0375	0.0354
40 x 3 x 6	0.0596	0.0700	0.0566	0.0456	0.0598	0.0407	0.0527	0.0488	0.0389
40 x 4 x 2	0.0840	-	0.0479	-	0.0439	-	0.0355	0.0294	0.0227
40 x 4 x 4	0.0858	-	0.0582	0.0579	0.0635	0.0517	0.0495	0.0435	0.0452
40 x 4 x 6	0.0764	0.0832	0.0627	0.0631	0.0685	0.0587	0.0640	0.0593	0.0554
50 x 2 x 2	0.0486	-	0.0175	-	0.0170	-	0.0098	0.0099	0.0126
50 x 2 x 4	0.0440	-	0.0299	-	0.0235	0.0214	0.0208	0.0188	0.0249
50 x 2 x 6	0.0455	0.0695	0.0337	0.0305	0.0357	0.0272	0.0315	0.0295	0.0285
50 x 3 x 2	0.0789	-	0.0303	-	0.0379	-	0.0269	0.0212	0.0197
50 x 3 x 4	0.0761	-	0.0523	-	0.0392	0.0482	0.0355	0.0353	0.0329
50 x 3 x 6	0.0702	-	0.0574	0.0456	0.0474	0.0381	0.0471	0.0433	0.0563
50 x 4 x 2	0.1070	-	0.0381	-	0.0451	-	0.0295	-	0.0209
50 x 4 x 4	0.0867	-	0.0666	-	0.0593	-	0.0533	0.0481	0.0424
50 x 4 x 6	0.0911	-	0.0746	0.0592	0.0607	0.0525	0.0597	0.0633	0.0600
Average *)	0.0621	-	0.0429	-	0.0423	-	0.0380	-	0.0343
Average **)	0.0553	0.0545	0.0453	0.0405	0.0441	0.0392	0.0382	0.0353	

*) = based on the average of all instances.

**) = based on the average of instances where TS-H2 gives a feasible solution.

Table 7.4: The CPU time (seconds) obtained using TS-H1 and TS-H2 (data set of type I).

	$b_\ell = 0$		$b_\ell = 1$		$b_\ell = ½\, U_\ell$		$b_\ell = U_\ell$		$b_\ell = \infty$
N x L x m_L	TS-H1	TS-H2	TS-H1	TS-H2	TS-H1	TS-H2	TS-H1	TS-H2	TS-Z1
20 x 2 x 2	2.36	2.72	3.69	3.76	3.86	3.57	4.33	3.01	0.46
20 x 2 x 4	2.17	2.82	3.28	4.42	3.13	4.59	2.57	3.87	0.61
20 x 2 x 6	3.14	4.64	3.15	3.47	3.52	3.85	3.11	5.02	0.65
20 x 3 x 2	5.48	4.87	6.51	7.39	6.10	5.29	7.27	5.86	0.68
20 x 3 x 4	5.14	7.62	6.67	8.86	6.17	8.91	6.83	7.06	0.83
20 x 3 x 6	6.86	7.09	7.19	8.67	5.25	6.70	6.97	8.32	1.08
20 x 4 x 2	8.70	8.39	10.55	10.31	11.56	9.59	11.28	8.85	1.23
20 x 4 x 4	10.24	13.18	8.77	9.46	10.06	9.84	12.34	10.47	1.30
20 x 4 x 6	7.49	7.80	8.67	6.97	8.27	7.04	9.97	8.42	1.54
30 x 2 x 2	8.05	-	9.44	10.91	10.40	10.21	11.59	10.63	1.68
30 x 2 x 4	7.67	12.04	8.35	10.01	13.73	13.57	10.61	14.27	1.57
30 x 2 x 6	7.26	10.40	10.14	13.94	10.32	19.98	8.16	16.71	2.09
30 x 3 x 2	18.26	-	19.61	19.98	19.27	23.62	20.94	23.53	2.38
30 x 3 x 4	18.08	16.20	20.96	25.71	17.44	22.98	19.90	31.91	3.09
30 x 3 x 6	18.12	19.91	16.77	40.00	17.99	30.01	22.43	26.07	3.15
30 x 4 x 2	32.71	-	38.99	-	48.27	44.10	44.39	42.96	2.62
30 x 4 x 4	24.35	28.59	35.58	40.12	25.26	34.86	30.72	50.06	3.80
30 x 4 x 6	29.11	31.75	30.51	47.59	33.41	48.68	27.39	43.49	6.00
40 x 2 x 2	17.20	-	26.17	-	22.03	-	36.60	24.93	2.68
40 x 2 x 4	14.45	-	25.58	23.72	23.89	26.29	28.54	32.09	2.98
40 x 2 x 6	17.14	23.87	14.56	28.18	18.87	35.93	23.31	42.72	4.30
40 x 3 x 2	32.57	-	56.92	-	49.94	-	54.94	69.54	5.13
40 x 3 x 4	34.88	-	39.89	41.51	40.51	69.10	55.01	59.58	6.86
40 x 3 x 6	48.31	40.56	51.14	62.44	48.76	75.45	51.87	52.41	7.75
40 x 4 x 2	70.52	-	85.15	-	77.96	-	125.76	101.00	7.76
40 x 4 x 4	44.68	-	59.32	95.84	57.79	116.37	85.64	153.46	9.39
40 x 4 x 6	63.39	70.68	63.10	78.90	74.38	81.33	67.35	108.77	8.77
50 x 2 x 2	33.04	-	43.25	-	33.87	-	74.00	94.50	4.14
50 x 2 x 4	27.50	-	33.68	-	41.99	58.94	51.23	102.44	6.76
50 x 2 x 6	26.08	44.58	39.13	53.42	32.21	71.73	41.84	51.12	6.33
50 x 3 x 2	86.84	-	89.98	-	96.70	-	108.23	175.81	10.48
50 x 3 x 4	58.20	-	80.65	-	95.21	82.42	100.39	176.88	15.73
50 x 3 x 6	53.54	-	70.64	137.08	80.86	152.78	93.89	108.86	11.52
50 x 4 x 2	128.35	-	203.30	-	153.78	-	235.78	-	15.23
50 x 4 x 4	149.06	-	144.86	-	147.35	-	165.77	253.47	23.03
50 x 4 x 6	114.55	-	104.59	305.15	141.38	210.97	122.22	274.88	18.43
Average *)	34.32	-	41.13	-	41.43	-	49.53	-	5.61
Average **)	16.37	18.83	26.07	42.22	31.37	44.44	44.21	62.94	

*) = based on the average of all instances.

**) = based on the average of instances where TS-H2 gives a feasible solution.

7.3.2 Data set of type II

In this category of instances, the optimal solutions are not known. Thus, our objective here is to study the performance of the algorithms as compared with the lower bound. As we discussed earlier, in this experiment, we run the program with $b_\ell = 0, b_\ell = 1, b_\ell = \frac{1}{2}U_\ell$, and $b_\ell = U_\ell$, for all $\ell = 1, ..., L-1$.

The experimental results are presented in Tables 7.5 to 7.10 which are shown in the similar format as the experimental results for data set of type I. In Tables 7.5 and 7.8, we present the makespans obtained for various buffer sizes for the cases of identical and non-identical number of machines at different stage, respectively. In these tables, we also present the results obtained using algorithm TS-Z1, i.e., for the associated unlimited buffer case. The corresponding performance ratio and the CPU time are shown in Tables 7.6 and 7.7, and in Tables 7.9 and 7.10, respectively. Similar with the previous case, TS-H2 fails to produce any feasible solution in some instances (as indicated in the table).

From these tables, we make the following observations.

1. We observe that the average performance ratio obtained using TS-H1 is getting better (smaller) as we increase the buffer capacity. For the case where the number of machines at each stage are identical (Table 7.6), the average performance ratio for $b_\ell = 0, 1, \frac{1}{2}U_\ell$, and U_ℓ, are 0.0697, 0.0378, 0.0344, and 0.0319, respectively. For the case where the number of machines at different stages are not identical (Table 7.9), the average performance ratio for

$b_\ell = 0, 1, \frac{1}{2}U_\ell$, and U_ℓ, are 0.0315, 0.0133, 0.0115, and 0.0083, respectively. This phenomenon is consistent with the phenomenon observed for data set of type I.

However in this case, since we do not know the optimal solution of each instance for a given buffer configuration, we are not able to make an objective conclusion with respect to the distance from the optimal makespan. As we mentioned earlier, in instances with smaller buffer capacities where the heuristic procedure finds a higher performance ratio, it could be either due to a correspondingly larger optimal makespan and/or due to the impact of the limited buffer capacities on the heuristic search itself.

Similar to the case for data set of type I, this phenomenon is not consistent among all instances. In some instances, we observe that the resulting makespan actually decreases as we reduce the buffer capacities. Indeed, in these instances, it is clear that this phenomenon is due to the favorable impact of the limited buffer capacities on the search procedure itself.

2. For the case of identical number of machines at all stages (Table 7.6), a comparison on the results obtained via TS-Z1 for the unlimited buffer sizes with the results of TS-H1 with $b_\ell = U_\ell$ shows a similar behavior as we observed in the case of data set of type I. In 11 out of 36 instances, TS-H1 obtains smaller makespan. The two different explanations that can be described for this observation are similar with those that we made for the

case of data set of type I. Likewise, we believe that this phenomenon is due to the limitation on the buffer sizes helps the search procedure.

3. For the case of non-identical number of machines (Table 7.9), we observe that for $b_\ell = U_\ell$, in 15 out of 20 instances TS-H1 yields smaller performance ratio than TS-Z1. The average ratio for TS-H1 is 0.83% above the lower bound. The corresponding value for TS-Z1 is 1.11%. These results show that for these instances, TS-H1 (with limited buffer capacities) seems to perform better than TS-Z1 (with unlimited buffer capacities). This pattern is not consistent with the previous two cases where we compared the results of TS-Z1 with those of TS-H1 (i.e., data sets of type I and II with identical number of machines at all stages).

   In order to better understand this phenomenon, we conducted a further experiment using the FAM rule in the context of the tabu search procedure for the unlimited buffer case. We refer to this algorithm as TS-FAM and we performed the experiment using the same data set (i.e., data set of type II with non-identical number of machines at different stages). The results are presented in Table 7.11, showing the makespan and the performance ratio obtained using TS-FAM. In this table, we also present the results obtained using TS-Z1. The results show that in 15 out of 20 instances, TS-FAM gives smaller makespan than TS-Z1. The average ratio obtained using TS-FAM (i.e., 0.86% above the lower bound) is also smaller than that obtained via

TS-Z1 (i.e., 1.11%). Based on these outcomes, we determine that in this case, the FAM rule is indeed better than procedure $Z1$.

Comparing the results obtained via TS-H1 with $b_\ell = U_\ell$ with those obtained via TS-FAM, we observe that in 12 out of 20 instances, TS-H1 gives smaller makespan than TS-FAM. In addition, TS-H1 with $b_\ell = U_\ell$ gives smaller average ratio than TS-FAM (0.0083 as compared with 0.0086). This result actually affirms our previous finding that the limitation on the buffer capacities can be helpful to the search procedure.

4. Comparing the results of Tables 7.3 and 7.6, we observe that the performance ratio for type I instances is generally larger than the corresponding ratio for the type II instances of similar sizes. Since in the former case the comparison is with the optimal makespan while in the latter case the comparison is with the lower bound, it follows that the heuristic procedure seems to perform better in the latter instances.

5. For the case of identical number of machines at all stages (Table 7.6), the performance ratio tends to increase as the number of stages increase and as the number of jobs increase. The number of machines at each stage (m_ℓ) on the performance ratio does not seem to have an effect that follows a particular pattern.

6. For the case of non-identical number of machines (Table 7.9), the effect of the

number of jobs and the number of stages do not seem to follow a particular pattern.

7. For the case of identical number of machines at all stages (Table 7.6), the number of instances in which TS-H2 fails to generate a feasible solution increases as the buffer capacities decrease. The highest number of failures occur when $b_\ell = 0$, for all ℓ. For the case of non-identical number of machines at different stages (Table 7.9), TS-H2 fails to generate a feasible solution in almost every instance when $b_\ell = 0, 1$, and $\frac{1}{2}U_\ell$.

8. Compared with TS-Z1, both TS-H1 and TS-H2 have larger CPU time, as expected.

From the above observations that we obtained from the experiment using data set of type I and data set of type II, we summarize the following.

- TS-H1, which is based on the FAM rule, tends to yield higher makespan as we decrease the buffer sizes. We have evidence that at least in some instances, this outcome is entirely due to the limitation imposed on the search procedure by the limited buffer capacities.

- The vector representation, with its implementation through TS-H1, is indeed an effective representation for solving FSPM/b problem. On the other hand, TS-H2 can fail to solve the problem as it did in many instances with smaller buffer capacities.

- As we discussed earlier, the challenge in the context of TS-H2 is how to generate a feasible schedule (I, X) for a given vector S (other than by using the FAM procedure, since it is already implemented in procedure $H1$). In both types of data set, in many instances we observe that our current implementation of TS-H2 fails to generate a feasible initial solution. This shows that for a given solution vector S, procedure $Z1$ is not an effective procedure to construct feasible I and X matrices which are needed in the implementation of procedure $H2$ for the FSPM/b problem. But, we also observe that in data set of type I, in those instances where TS-H2 terminates with a feasible solution, the average results obtained are smaller than those obtained using TS-H1. This observation compel us to continue our search for devising an effective constructive procedure to replace procedure $Z1$ in the context of TS-H2. Devising such a procedure which guarantees that the solution obtained is feasible for the FSPM/b problem is a subject of future research.

- Recall that we use the lower bounds developed for the FSPM problem in order to measure the solution quality. From the experimental results we determine that these lower bounds are relatively strong for data set of type II of the FSPM/b problem.

- For solving the FSPM problem, our preliminary investigation (as discussed in section 4.1.2) for the case of identical number of machines at all stages shows

that in the context of the tabu search, procedure $Z1$ performs better than the FAM rule. However, for the case of non-identical number of machines at different stages, we may consider the FAM rule as a viable alternative.

7.3.3 Data set of type III (Wittrock test instances)

In this part of the study, we conduct an experiment using the six test instances proposed by Wittrock [66] as discussed in section 7.1.3. To solve these instances, we use algorithm TS-Z1 for the case of unlimited buffer sizes, and algorithm TS-H1 for the limited buffer case.

In this experiment, we compare the makespan and the performance ratio obtained using our proposed algorithms with those obtained using algorithms WLA by Wittrock [66] and RITM by Sawik [56]. Both these algorithms use constructive heuristic methods and we quote the results from the respective papers. (Notice that we use the same lower bounds as the basis to measure the performance ratio of the three methods).

The results of the experiment for the unlimited buffer case are shown in Table 7.12. The first column indicates the instance number and second column indicates the best lower bound for each instance. In the following columns, we present the makespan (C_h), the performance ratio (η_h), and the number of buffer spaces utilized between stages 1 and 2 (U_1) and between stages 2 and 3 (U_2) as obtained using algorithm TS-Z1. As for comparison, we present the makespan (C_h), the

Table 7.5: The makespan obtained using TS-H1 and TS-H2 (data set of type II with identical number of machines at all stages).

	$b_\ell = 0$		$b_\ell = 1$		$b_\ell = ½\,U_\ell$		$b_\ell = U_\ell$		$b_\ell = \infty$
N x L x m_L	TS-H1	TS-H2	TS-H1	TS-H2	TS-H1	TS-H2	TS-H1	TS-H2	TS-Z1
20 x 2 x 2	557.60	-	552.20	552.60	550.00	550.00	550.80	550.20	550.60
20 x 2 x 4	290.00	292.40	283.40	283.40	283.80	284.40	281.80	284.80	282.20
20 x 2 x 6	224.20	223.60	222.20	223.60	224.40	224.20	225.00	223.60	222.60
20 x 3 x 2	644.00	-	614.80	622.40	616.40	620.80	612.00	614.40	613.20
20 x 3 x 4	340.20	344.40	337.00	334.80	337.00	339.40	336.00	334.20	333.60
20 x 3 x 6	259.60	260.20	259.80	259.40	259.60	260.20	259.20	259.40	259.40
20 x 4 x 2	685.60	-	639.60	648.60	646.80	652.60	638.20	637.20	637.20
20 x 4 x 4	406.20	408.00	400.20	404.00	407.20	403.00	401.80	398.40	400.20
20 x 4 x 6	316.80	315.40	316.00	316.60	315.60	317.60	315.60	318.80	313.60
30 x 2 x 2	775.00	-	748.40	753.80	748.20	751.20	747.60	746.60	747.40
30 x 2 x 4	424.80	437.60	417.60	419.00	417.00	415.80	414.60	414.80	415.20
30 x 2 x 6	275.20	279.40	271.00	270.60	271.20	270.80	268.80	269.80	269.00
30 x 3 x 2	895.80	-	836.80	-	833.40	836.80	834.00	836.80	834.00
30 x 3 x 4	483.80	500.00	476.20	476.20	478.40	476.00	471.00	469.60	469.60
30 x 3 x 6	319.00	319.80	315.20	313.80	314.20	313.20	313.80	311.00	312.00
30 x 4 x 2	980.60	-	910.20	-	902.80	922.60	893.00	897.40	901.40
30 x 4 x 4	499.40	517.20	487.00	489.80	485.60	488.20	477.60	478.20	478.40
30 x 4 x 6	385.00	387.80	385.40	385.20	384.20	386.20	382.00	380.80	381.80
40 x 2 x 2	1092.60	-	1061.00	-	1060.00	1061.40	1059.80	1059.80	1058.80
40 x 2 x 4	568.80	593.60	559.60	567.00	558.00	558.80	557.80	557.80	556.00
40 x 2 x 6	369.40	375.80	365.20	370.20	365.20	363.40	365.80	363.00	363.00
40 x 3 x 2	1151.60	-	1095.20	-	1087.00	1088.40	1090.00	1089.20	1084.20
40 x 3 x 4	593.40	614.20	573.20	578.40	568.60	571.20	571.40	567.20	566.80
40 x 3 x 6	408.00	426.40	408.40	406.20	403.00	399.60	401.60	397.40	397.00
40 x 4 x 2	1239.00	-	1126.60	-	1112.00	-	1119.00	1114.60	1113.00
40 x 4 x 4	657.00	-	629.20	649.00	625.80	629.00	624.80	620.20	615.80
40 x 4 x 6	471.00	482.40	462.20	468.00	465.80	463.60	462.20	459.40	459.80
50 x 2 x 2	1369.40	-	1332.00	-	1327.00	1327.60	1328.60	1327.00	1325.80
50 x 2 x 4	678.80	-	661.20	-	658.00	660.40	659.20	658.80	662.80
50 x 2 x 6	464.20	472.80	456.20	457.60	452.80	453.80	454.20	452.20	454.00
50 x 3 x 2	1493.80	-	1406.20	-	1385.80	1394.80	1388.00	1394.40	1393.80
50 x 3 x 4	726.40	-	695.20	-	692.00	689.40	693.80	688.40	689.60
50 x 3 x 6	517.60	529.60	503.60	509.40	500.60	503.20	498.00	496.80	500.20
50 x 4 x 2	1566.40	-	1441.20	-	1409.00	-	1406.40	1400.00	1408.20
50 x 4 x 4	775.60	-	748.20	-	733.60	738.00	740.60	730.80	737.00
50 x 4 x 6	556.00	571.00	551.80	550.20	546.80	546.60	548.20	549.60	547.00
Average *)	651.72	-	626.37	-	622.97	-	622.01	620.91	620.95
Average **)	408.63	417.58	449.42	452.39	585.46	587.12	622.01	620.91	

*) = based on the average of all instances.

**) = based on the average of instances where TS-H2 gives a feasible solution.

Table 7.6: The performance ratio obtained using TS-H1 and TS-H2 (data set of type II with identical number of machines at all stages).

	$b_\ell = 0$		$b_\ell = 1$		$b_\ell = ½\, U_\ell$		$b_\ell = U_\ell$		$b_\ell = \infty$
N x L x m_L	TS-H1	TS-H2	TS-H1	TS-H2	TS-H1	TS-H2	TS-H1	TS-H2	TS-Z1
20 x 2 x 2	0.0145	-	0.0047	0.0049	0.0004	0.0004	0.0019	0.0007	0.0015
20 x 2 x 4	0.0478	0.0561	0.0241	0.0244	0.0252	0.0274	0.0182	0.0289	0.0199
20 x 2 x 6	0.0872	0.0846	0.0779	0.0849	0.0888	0.0878	0.0914	0.0849	0.0800
20 x 3 x 2	0.0585	-	0.0079	0.0211	0.0108	0.0182	0.0029	0.0073	0.0049
20 x 3 x 4	0.1024	0.1178	0.0916	0.0847	0.0915	0.0995	0.0883	0.0829	0.0811
20 x 3 x 6	0.0152	0.0174	0.0158	0.0144	0.0154	0.0174	0.0137	0.0147	0.0144
20 x 4 x 2	0.1056	-	0.0318	0.0462	0.0430	0.0522	0.0296	0.0278	0.0281
20 x 4 x 4	0.1188	0.1239	0.1026	0.1133	0.1215	0.1104	0.1072	0.0977	0.1024
20 x 4 x 6	0.1035	0.0984	0.1004	0.1026	0.0992	0.1059	0.0992	0.1113	0.0917
30 x 2 x 2	0.0384	-	0.0033	0.0103	0.0029	0.0068	0.0022	0.0009	0.0019
30 x 2 x 4	0.0309	0.0613	0.0130	0.0164	0.0112	0.0083	0.0052	0.0056	0.0066
30 x 2 x 6	0.0513	0.0660	0.0337	0.0341	0.0359	0.0341	0.0269	0.0311	0.0276
30 x 3 x 2	0.0793	-	0.0080	-	0.0039	0.0079	0.0048	0.0083	0.0046
30 x 3 x 4	0.0638	0.0998	0.0468	0.0471	0.0517	0.0467	0.0354	0.0322	0.0323
30 x 3 x 6	0.0730	0.0760	0.0601	0.0556	0.0568	0.0537	0.0558	0.0459	0.0495
30 x 4 x 2	0.1151	-	0.0338	-	0.0252	0.0482	0.0137	0.0188	0.0231
30 x 4 x 4	0.1033	0.1419	0.0758	0.0818	0.0726	0.0778	0.0550	0.0563	0.0570
30 x 4 x 6	0.0897	0.0976	0.0908	0.0904	0.0872	0.0933	0.0811	0.0777	0.0805
40 x 2 x 2	0.0330	-	0.0028	-	0.0018	0.0031	0.0017	0.0016	0.0007
40 x 2 x 4	0.0269	0.0713	0.0096	0.0232	0.0067	0.0083	0.0063	0.0065	0.0033
40 x 2 x 6	0.0294	0.0476	0.0179	0.0317	0.0178	0.0128	0.0193	0.0116	0.0116
40 x 3 x 2	0.0641	-	0.0118	-	0.0041	0.0054	0.0066	0.0059	0.0014
40 x 3 x 4	0.0612	0.0987	0.0246	0.0336	0.0160	0.0208	0.0214	0.0135	0.0128
40 x 3 x 6	0.0572	0.1056	0.0586	0.0525	0.0444	0.0352	0.0405	0.0295	0.0286
40 x 4 x 2	0.1263	-	0.0232	-	0.0099	-	0.0163	0.0121	0.0105
40 x 4 x 4	0.0990	-	0.0520	0.0853	0.0460	0.0517	0.0449	0.0369	0.0297
40 x 4 x 6	0.0906	0.1179	0.0706	0.0840	0.0784	0.0739	0.0703	0.0640	0.0649
50 x 2 x 2	0.0339	-	0.0059	-	0.0021	0.0025	0.0034	0.0021	0.0012
50 x 2 x 4	0.0373	-	0.0101	-	0.0049	0.0086	0.0068	0.0061	0.0123
50 x 2 x 6	0.0371	0.0562	0.0189	0.0222	0.0114	0.0135	0.0149	0.0100	0.0139
50 x 3 x 2	0.0830	-	0.0178	-	0.0032	0.0095	0.0044	0.0096	0.0088
50 x 3 x 4	0.0693	-	0.0227	-	0.0175	0.0138	0.0200	0.0123	0.0140
50 x 3 x 6	0.0688	0.0937	0.0395	0.0519	0.0335	0.0389	0.0281	0.0256	0.0324
50 x 4 x 2	0.1280	-	0.0364	-	0.0123	-	0.0109	0.0061	0.0122
50 x 4 x 4	0.0939	-	0.0549	-	0.0334	0.0400	0.0436	0.0296	0.0389
50 x 4 x 6	0.0704	0.0998	0.0624	0.0593	0.0529	0.0523	0.0556	0.0580	0.0531
Average *)	0.0697	-	0.0378	-	0.0344	-	0.0319	0.0298	0.0294
Average **)	0.0664	0.0866	0.0454	0.0510	0.0358	0.0378	0.0319	0.0298	

*) = based on the average of all instances.

**) = based on the average of instances where TS-H2 gives a feasible solution.

Table 7.7: The CPU time (seconds) obtained using TS-H1 and TS-H2 (data set of type II with identical number of machines at all stages).

	$b_\ell = 0$		$b_\ell = 1$		$b_\ell = ½\, U_\ell$		$b_\ell = U_\ell$		$b_\ell = \infty$
N x L x m_L	TS-H1	TS-H2	TS-H1	TS-H2	TS-H1	TS-H2	TS-H1	TS-H2	TS-Z1
20 x 2 x 2	2.45	-	2.76	2.42	3.03	3.46	2.64	2.75	0.33
20 x 2 x 4	2.67	3.47	4.13	3.36	3.51	3.68	3.28	3.56	0.63
20 x 2 x 6	3.89	5.12	3.96	4.71	3.41	4.40	2.84	4.90	0.72
20 x 3 x 2	6.42	-	5.00	4.90	6.16	5.28	5.72	5.43	0.88
20 x 3 x 4	6.55	7.13	5.85	9.30	6.26	6.50	5.13	6.69	0.84
20 x 3 x 6	5.47	5.02	5.88	5.76	5.14	5.47	5.16	5.56	0.79
20 x 4 x 2	7.45	-	13.36	11.23	8.25	10.40	9.56	10.72	1.27
20 x 4 x 4	8.36	12.96	13.43	8.88	9.09	10.74	10.96	10.74	1.66
20 x 4 x 6	9.30	12.74	8.27	9.01	11.24	10.18	10.27	10.65	1.43
30 x 2 x 2	8.18	-	8.38	10.13	7.12	7.74	6.43	10.36	0.86
30 x 2 x 4	8.48	7.66	8.42	12.02	7.36	14.04	7.74	11.51	1.22
30 x 2 x 6	10.37	9.71	11.40	11.76	11.06	13.43	13.81	13.85	2.28
30 x 3 x 2	11.91	-	15.75	-	17.75	14.87	15.29	13.27	1.78
30 x 3 x 4	19.67	16.87	18.92	24.64	17.11	24.37	21.15	26.87	2.68
30 x 3 x 6	19.34	24.69	24.82	30.80	21.56	24.64	24.76	34.77	3.24
30 x 4 x 2	24.86	-	31.15	-	30.97	37.55	34.13	37.79	2.89
30 x 4 x 4	29.64	25.35	31.77	38.43	27.26	28.97	34.08	37.21	4.54
30 x 4 x 6	33.13	42.26	34.70	36.47	30.90	36.47	36.53	34.49	4.24
40 x 2 x 2	12.42	-	14.08	-	12.22	17.62	12.86	17.88	1.72
40 x 2 x 4	17.54	20.15	18.75	21.40	17.38	33.97	19.52	24.39	2.75
40 x 2 x 6	25.89	28.38	28.47	26.48	21.06	36.56	23.03	37.65	3.68
40 x 3 x 2	34.13	-	34.34	-	32.09	45.33	27.63	39.93	3.68
40 x 3 x 4	36.27	32.98	38.32	52.95	38.52	71.04	37.04	48.38	5.22
40 x 3 x 6	50.07	44.11	45.16	52.63	66.03	69.81	52.46	66.32	8.79
40 x 4 x 2	62.42	-	81.34	-	62.88	-	65.87	71.14	6.25
40 x 4 x 4	58.45	-	91.13	59.99	80.02	111.16	63.11	91.86	9.90
40 x 4 x 6	85.85	95.05	78.29	102.79	70.32	87.80	78.83	85.16	8.86
50 x 2 x 2	24.40	-	24.93	-	24.77	32.84	22.94	32.08	2.63
50 x 2 x 4	30.77	-	37.44	-	32.53	39.93	31.13	40.84	4.29
50 x 2 x 6	33.26	53.22	34.82	54.97	50.04	58.18	53.57	62.25	6.51
50 x 3 x 2	61.36	-	58.01	-	71.07	67.88	54.21	77.53	4.14
50 x 3 x 4	70.35	-	86.88	-	63.21	100.51	68.59	109.27	8.46
50 x 3 x 6	82.50	95.07	83.92	104.04	121.01	106.25	107.93	148.12	12.08
50 x 4 x 2	89.42	-	138.01	-	122.18	-	118.91	141.20	8.57
50 x 4 x 4	135.83	-	137.98	-	164.28	181.22	148.78	185.17	16.96
50 x 4 x 6	135.78	168.51	140.90	234.02	178.20	213.52	171.06	164.35	18.21
Average *)	35.13	-	39.46	-	40.42	-	39.08	47.91	4.58
Average **)	31.20	35.52	30.43	37.32	37.35	45.17	39.08	47.91	

*) = based on the average of all instances.

**) = based on the average of instances where TS-H2 gives a feasible solution.

Table 7.8: The makespan obtained using TS-H1 and TS-H2 (data set of type II with non-identical number of machines at different stages).

			$b_\ell = 0$		$b_\ell = 1$		$b_\ell = ½\ U_\ell$		$b_\ell = U_\ell$		$b_\ell = \infty$
N	x	L	TS-H1	TS-H2	TS-H1	TS-H2	TS-H1	TS-H2	TS-H1	TS-H2	TS-Z1
20	x	2	619.40	-	615.60	-	615.90	616.20	615.10	614.80	614.80
20	x	3	687.10	-	678.50	-	679.50	-	679.30	679.70	682.70
20	x	4	871.90	-	856.50	-	870.60	-	855.20	856.30	855.30
20	x	6	891.10	-	874.80	-	876.40	-	871.10	872.70	879.70
20	x	8	1176.70	-	1155.90	-	1150.30	-	1144.70	1154.70	1151.10
30	x	2	1029.90	-	1028.80	-	1029.20	-	1029.20	1028.80	1028.50
30	x	3	1165.60	-	1148.30	-	1149.30	-	1146.70	1149.00	1157.70
30	x	4	1139.50	-	1113.90	-	1109.70	-	1106.70	1108.90	1108.20
30	x	6	1500.40	-	1470.00	-	1468.50	-	1463.00	1474.20	1473.90
30	x	8	1832.50	-	1784.10	-	1775.50	-	1765.50	1788.40	1781.70
40	x	2	1135.80	-	1128.90	-	1128.10	-	1128.40	1128.20	1128.90
40	x	3	1368.40	-	1350.50	-	1351.70	-	1349.80	1349.00	1352.70
40	x	4	1507.20	-	1498.20	-	1498.70	-	1498.00	1500.00	1500.40
40	x	6	1873.60	-	1832.20	-	1823.70	-	1821.00	1828.30	1823.30
40	x	8	2270.50	-	2193.70	-	2167.00	-	2154.80	-	2164.50
50	x	2	1211.10	-	1203.90	-	1203.60	-	1204.30	1203.90	1203.30
50	x	3	1532.20	-	1499.40	-	1487.10	-	1484.80	1488.50	1486.60
50	x	4	1728.00	-	1690.60	-	1688.80	-	1690.90	1692.30	1685.70
50	x	6	2609.10	-	2592.50	-	2603.20	-	2596.10	2601.10	2604.90
50	x	8	2642.80	-	2535.10	-	2504.30	-	2489.20	2506.20	2515.90
Average			1439.64	-	1412.57	-	1409.06	-	1404.69	-	1409.99

Table 7.9: Deviation (η_h) from the lower bound obtained using TS-H1 and TS-H2 (data set of type II with non-identical number of machines at different stages).

			$b_\ell = 0$		$b_\ell = 1$		$b_\ell = ½\, U_\ell$		$b_\ell = U_\ell$		$b_\ell = \infty$
N	x	L	TS-H1	TS-H2	TS-H1	TS-H2	TS-H1	TS-H2	TS-H1	TS-H2	TS-Z1
20	x	2	0.0173	-	0.0066	-	0.0077	0.0087	0.0055	0.0042	0.0043
20	x	3	0.0207	-	0.0072	-	0.0095	-	0.0089	0.0091	0.0135
20	x	4	0.0433	-	0.0217	-	0.0323	-	0.0175	0.0172	0.0170
20	x	6	0.0480	-	0.0360	-	0.0352	-	0.0295	0.0291	0.0360
20	x	8	0.0475	-	0.0303	-	0.0246	-	0.0192	0.0265	0.0237
30	x	2	0.0053	-	0.0020	-	0.0030	-	0.0027	0.0022	0.0012
30	x	3	0.0205	-	0.0055	-	0.0060	-	0.0030	0.0053	0.0110
30	x	4	0.0353	-	0.0111	-	0.0058	-	0.0029	0.0042	0.0034
30	x	6	0.0338	-	0.0127	-	0.0118	-	0.0080	0.0154	0.0163
30	x	8	0.0399	-	0.0134	-	0.0087	-	0.0030	0.0159	0.0121
40	x	2	0.0148	-	0.0052	-	0.0044	-	0.0045	0.0044	0.0057
40	x	3	0.0187	-	0.0031	-	0.0053	-	0.0023	0.0019	0.0058
40	x	4	0.0122	-	0.0035	-	0.0040	-	0.0038	0.0035	0.0040
40	x	6	0.0437	-	0.0190	-	0.0125	-	0.0114	0.0159	0.0122
40	x	8	0.0610	-	0.0249	-	0.0119	-	0.0061	-	0.0103
50	x	2	0.0136	-	0.0042	-	0.0037	-	0.0041	0.0045	0.0031
50	x	3	0.0373	-	0.0153	-	0.0054	-	0.0037	0.0062	0.0050
50	x	4	0.0355	-	0.0147	-	0.0134	-	0.0163	0.0144	0.0097
50	x	6	0.0082	-	0.0018	-	0.0060	-	0.0031	0.0054	0.0067
50	x	8	0.0728	-	0.0286	-	0.0177	-	0.0113	0.0192	0.0215
Average			0.0315	-	0.0133	-	0.0115	-	0.0083	-	0.0111

Table 7.10: The CPU time (seconds) obtained using TS-H1 and TS-H2 (data set of type II with non-identical number of machines at different stages).

			$b_\ell = 0$		$b_\ell = 1$		$b_\ell = ½\, U_\ell$		$b_\ell = U_\ell$		$b_\ell = \infty$
N	x	L	TS-H1	TS-H2	TS-H1	TS-H2	TS-H1	TS-H2	TS-H1	TS-H2	TS-Z1
20	x	2	2.57	-	3.18	-	3.00	3.02	2.63	3.1	0.52
20	x	3	4.89	-	5.40	-	5.73	-	4.26	7.0	0.71
20	x	4	7.88	-	9.63	-	8.47	-	7.70	10.0	1.02
20	x	6	17.23	-	16.84	-	13.94	-	18.98	20.8	1.65
20	x	8	24.00	-	27.68	-	29.07	-	28.68	27.0	1.92
30	x	2	6.41	-	7.27	-	6.90	-	7.52	9.6	1.13
30	x	3	16.85	-	16.46	-	15.00	-	14.79	18.5	1.75
30	x	4	21.77	-	24.55	-	24.73	-	22.52	29.0	2.80
30	x	6	54.13	-	60.64	-	53.39	-	48.15	55.4	3.49
30	x	8	80.97	-	87.40	-	75.36	-	85.87	89.4	4.58
40	x	2	12.56	-	19.13	-	15.24	-	13.36	19.4	2.03
40	x	3	27.38	-	37.60	-	28.39	-	29.84	35.8	2.76
40	x	4	47.38	-	56.30	-	46.06	-	43.78	56.5	3.83
40	x	6	99.71	-	100.36	-	99.87	-	105.03	92.3	6.85
40	x	8	178.82	-	182.28	-	189.09	-	201.84	-	10.43
50	x	2	27.02	-	30.19	-	26.84	-	24.45	36.5	4.00
50	x	3	62.10	-	57.35	-	68.13	-	52.22	67.8	6.43
50	x	4	96.20	-	95.71	-	91.69	-	91.30	101.7	8.61
50	x	6	178.66	-	167.73	-	188.04	-	300.64	178.4	8.41
50	x	8	387.39	-	404.36	-	359.72	-	341.78	347.2	13.37
Average			67.70	-	70.50	-	67.43	-	72.27	63.44	4.31

Table 7.11: The makespan (C_h) and the performance ratio (η_h) obtained using TS-FAM and TS-Z1 for b_ℓ = unlimited (data set of type II with non-identical number of machines at different stages).

			C_h		η_h	
N	x	L	TS-FAM	TS-Z1	TS-FAM	TS-Z1
20	x	2	615.50	614.80	0.0061	0.0043
20	x	3	679.00	682.70	0.0080	0.0135
20	x	4	854.50	855.30	0.0158	0.0170
20	x	6	872.30	879.70	0.0305	0.0360
20	x	8	1147.40	1151.10	0.0217	0.0237
30	x	2	1029.50	1028.50	0.0034	0.0012
30	x	3	1147.00	1157.70	0.0034	0.0110
30	x	4	1107.20	1108.20	0.0041	0.0034
30	x	6	1464.70	1473.90	0.0102	0.0163
30	x	8	1766.10	1781.70	0.0034	0.0121
40	x	2	1128.20	1128.90	0.0044	0.0057
40	x	3	1349.20	1352.70	0.0019	0.0058
40	x	4	1496.90	1500.40	0.0024	0.0040
40	x	6	1823.80	1823.30	0.0124	0.0122
40	x	8	2156.70	2164.50	0.0069	0.0103
50	x	2	1203.60	1203.30	0.0039	0.0031
50	x	3	1483.70	1486.60	0.0024	0.0050
50	x	4	1686.90	1685.70	0.0111	0.0097
50	x	6	2601.50	2604.90	0.0051	0.0067
50	x	8	2498.60	2515.90	0.0146	0.0215
Average			1405.62	1409.99	0.0086	0.0111

performance ratio (η_h), and the *maximum queue* obtained using algorithm WLA by Wittrock [66]. Recall that for this case, Wittrock allocates an unlimited number of buffer spaces in front of each machine; the term *maximum queue* refers to the maximum number of jobs waiting in front of a machine, either at stage 2 or at stage 3. In our experiment, the number of buffer spaces utilized U_ℓ refers to the maximum number of jobs waiting in the buffer area following stage ℓ.

From this table, we make the following observations.

- In 5 of the 6 instances, TS-Z1 gives lower makespan than WLA.
- The average performance ratio obtained using TS-Z1 is 1.91% above the lower bound which is smaller than the value obtained using WLA with an average value of 3.53%.

For the limited buffer case with $b_\ell = \{3, 3\}$, we present our computational results with TS-H1 in Table 7.13. In this table, we also present the computational results obtained using algorithm WLA by Wittrock [66], as well as those obtained via algorithm RITM by Sawik [56]. (Recall that in this case, Wittrock and Sawik allocate one buffer space in front of each machine. Thus, there are three buffer spaces allocated between stages 1 and 2 and between stages 2 and 3). From this table, we make the following observations.

- In 5 of the 6 instances, TS-H1 obtains a solution with a significantly smaller makespan as compared with the other algorithms.

- The average performance ratio obtained using TS-H1 is 2.19% above the lower bound. This value is much smaller than the values obtained by either WLA or RITM. The corresponding values for WLA and RITM are 15.67% and 10.74%, respectively.

It is also of interest to compare the performance of algorithms TS-H1 and TS-H2 with each other in solving the Wittrock test instances. For this purpose, we conduct an experiment using these algorithms for $b_\ell = \{3, 3\}$ and $b_\ell = \{0, 0\}$. The results are shown in Table 7.14. The corresponding CPU times and the maximum buffer utilization are shown in Table 7.15. From these tables, we make the following observations.

- For $b_\ell = \{3, 3\}$, we can see that TS-H2 gives a feasible solution in all instances. However, the performance of TS-H2 with an average performance ratio 2.57% is worse than the performance of TS-H1.

- For $b_\ell = \{0, 0\}$, TS-H1 gives an average performance ratio of 7.1% above the lower bound. Observing the results obtained, TS-H2 fails to generate a feasible initial solution for instance number 1. In those instances where TS-H2 yields a feasible solution, TS-H1 with an average performance ratio 6.49% performs better than TS-H2 with an average performance ratio 11.19%.

From all of the above observations, we summarize the following.

Table 7.12: Experimental results using Wittrock test instances (unlimited buffer sizes).

Problem	Bound LB	TS-Z1					WLA		
		C_h	η_h	Cpu seconds	U_1	U_2	C_h	η_h	Max Queue
1	746.3	764	0.0237	4.34	12	3	784	0.0505	3
2	758.0	767	0.0119	5.16	6	4	789	0.0409	3
3	758.7	771	0.0163	3.84	6	4	785	0.0347	2
4	755.3	784	0.0380	3.24	5	4	796	0.0538	3
5	961.5	969	0.0078	2.04	2	1	964	0.0026	1
6	666.7	678	0.0170	1.54	6	6	686	0.0290	2
		average:	0.0191	3.36			average:	0.0353	

- For the unlimited buffer case, TS-Z1 performs better than algorithm WLA, and for the limited buffer case, TS-H1 performs better than algorithms WLA and RITM. As expected, the corresponding running time of TS-Z1 and TS-H1 tend to be higher than those reported in [66] for WLA and in [56] for RITM (although a direct comparison is not appropriate since we use different computer platform).

- As in the previous data sets, TS-H2 fails to solve some of the instances with limited buffer capacities. In those instances where TS-H2 finds a feasible solutions, still TS-H1 performs better than TS-H2.

Table 7.13: Experimental results using Wittrock test instances (limited buffer sizes, $b_\ell = \{3, 3\}$).

Problem	Bound LB	TS-H1				WLA		RITM	
		C_h	η_h	U_1	U_2	C_h	η_h	C_h	η_h
1	746.3	776	0.0398	3	3	907	0.2153	837	0.1215
2	758.0	774	0.0211	3	3	972	0.2823	870	0.1478
3	758.7	777	0.0242	3	3	885	0.1665	838	0.1046
4	755.3	775	0.0260	3	3	940	0.2445	922	0.2207
5	961.5	969	0.0078	2	1	964	0.0026	999	0.0390
6	666.7	675	0.0125	3	3	686	0.0290	674	0.0110
		average:	0.0219			average:	0.1567	average:	0.1074

Table 7.14: The makespan and the performance ratio obtained using TS-H1 and TS-H2 on the Wittrock test instances.

Problem	Bound LB	TS-H1		TS-H2	
		C_h	η_h	C_h	η_h
			b_ℓ =	{3,3}	
1	746.3	776	0.0398	771	0.0331
2	758.0	774	0.0211	767	0.0119
3	758.7	777	0.0242	775	0.0215
4	755.3	775	0.0260	804	0.0644
5	961.5	969	0.0078	969	0.0078
6	666.7	675	0.0125	677	0.0155
		average:	0.0219	average:	0.0257
			b_ℓ =	{0,0}	
1	746.3	822	0.1014	-	-
2	758.0	839	0.1069	879	0.1596
3	758.7	822	0.0835	892	0.1757
4	755.3	825	0.0922	877	0.1611
5	961.5	974	0.0130	977	0.0161
6	666.7	686	0.0290	698	0.0470
		average:	0.0710	average:	0.1119

Table 7.15: The CPU time (seconds) and the maximum buffer utilization obtained using TS-H1 and TS-H2 on the Wittrock test instances.

Problem	TS-H1			TS-H2		
	U_1	U_2	CPU secs.	U_1	U_2	CPU secs.
	b_ℓ = {3,3}					
1	3	3	121.17	3	3	180.87
2	3	3	44.81	3	3	67.56
3	3	3	32.52	3	3	36.47
4	3	3	24.72	3	2	19.88
5	2	1	19.28	2	1	19.99
6	3	3	13.73	3	3	14.50
	average:		42.71	average:		56.55
	b_ℓ = {0,0}					
1	0	0	72.89	-	-	-
2	0	0	37.02	0	0	39.33
3	0	0	25.76	0	0	20.92
4	0	0	48.39	0	0	35.54
5	0	0	30.27	0	0	21.20
6	0	0	11.26	0	0	13.41
	average:		37.60	average:		26.08

Chapter 8

Summary, Conclusions, and Future Research

We investigated the problem of scheduling N jobs on identical parallel machines at L stages. A previous study has shown that this problem is NP-hard. We considered two versions of this problem, namely the FSPM and the FSPM/b problem, with the objective of minimizing the makespan.

A literature review shows that there has been little contribution in this field, especially for the FSPM/b problem. Our objective in this investigation was to develop appropriate procedures for solving this problem and investigate the effectiveness of various search methods in this context.

8.1 Summary of the results

We proposed several search procedures for solving the FSPM problem which are based on a vector representation of the solution. This approach is in sharp contrast to a previously developed approach which is based on the matrix representation of a solution. We also proposed a new search procedure for the problem based on the matrix representation of the solution which is an adaptation of the approach proposed by Nowicki et al. [44].

Following is the summary of the results that we have obtained concerning the FSPM problem.

1. In the context of the vector representation of the solution, we developed two procedures, namely procedures $Z1$ and $Z2$, to construct a complete schedule for the FSPM problem for a given vector representation. Procedure $Z1$ is inspired by the algorithm by J. Carlier [8] which is developed for the 1-stage parallel machine problem with m identical machines with *release dates* and *tails*. Procedure $Z2$ is a variation of the FAM rule.

2. Using the vector representation and the two procedures $Z1$ and $Z2$, we developed three search procedures for the FSPM problem, i.e., a local improvement procedure, a genetic algorithm, and a tabu search.

3. We designed and developed a class of randomly generated problem instances for which the corresponding optimal makespans are known (data set of type

I). We also adopted a set of randomly generated problem instances from the open literature (data set of type II).

4. To measure the solution quality of the instances where their optimal solution are not known, we developed a set of lower bounds. We showed that these lower bounds dominate most of the existing lower bounds in the open literature.

5. Using the proposed procedures and the two types of data sets, we performed a comprehensive computational experiment. We compared the results obtained via the heuristic procedures with the corresponding optimal values and lower bounds for the two types of data set, respectively. We also compared the results obtained with the results available in the open literature which are solved using matrix representation.

In the context of FSPM/b problem, we determined that the presence of limited buffer capacities adds to the degree of complexity of the problem. Particularly in the manner of constructing a complete schedule for a given solution vector. As a result, none of the procedures that we introduced for the FSPM problem are applicable. Thus, to construct a complete schedule for a given solution, our approach is to establish the methods to simulate the flow of the jobs throughout the stages.

The following describes the results that we obtained for the FSPM/b problem.

1. We developed two procedures to construct a complete schedule for a given solution vector, namely procedures $H1$ and $H2$. Procedure $H1$ builds the complete schedule directly from the given solution vector S. On the other hand, procedure $H2$ requires to construct a complete schedule (I, X) for the given vector, before evaluating its corresponding makespan. Thus, procedure $H2$ is a generic procedure for constructing a complete schedule for the FSPM/b problem. In our study, we employed procedure $Z1$ to generate the I and X matrices in this context.

2. We developed the method to determine whether a given solution (I, X) is feasible (in the context of procedure $H2$).

3. We implemented the tabu search procedure for the FSPM/b problem based on vector representation with both procedures $H1$ and $H2$.

4. For the computational experiment, we generated a class of problem instances for which the corresponding optimal makespans are known (data set of type I) and we adopted a set of randomly generated problem instances from the open literature (data set of type II). In addition, we also employed a collection of test instances from a real production line, which is available in the open literature.

5. Using the tabu search procedure, we performed an experiment to study the effectiveness of procedures $H1$ and $H2$ in solving the FSPM/b problem. To

evaluate the solution quality, for data set of type I, we compared the results obtained with the optimal makespan. For data set of type II and the instances from the open literature, we evaluate the deviation of the makespan obtained from the lower bound developed for the FSPM problem.

6. In the context of FSPM problem, we also made a comparison on the performance of tabu search with the FAM rule (i.e., TS-FAM) with TS-Z1.

8.2 Conclusions

Based on the empirical analysis, we determined that among the proposed algorithms, the tabu search procedure with procedure $Z1$ to construct a complete procedure, i.e., TS-Z1, is the most effective procedure. An evaluation on the solution quality of the results obtained via TS-Z1 with respect to some reference values has shown that TS-Z1 seems to be quite effective in solving the FSPM problem. TS-Z1 also compares well with the heuristics available in the open literature which are based on the constructive method and the search method using matrix representation. In addition, we determined that the vector representation has an advantage in which it is simpler to work with, especially for solving the FSPM/b problem.

A separate study using TS-Z1 and TS-FAM for solving the FSPM problem shows that TS-Z1 performs better than TS-FAM for the case of identical number of machines at all stages. But for the case of non-identical number of machines at

different stages, TS-FAM can be considered as a viable alternative.

In the context of FSPM/b problem, we determined that the vector representation is an effective representation for solving the FSPM/b problem, particularly with its implementation through the FAM rule, i.e., procedure $H1$. The experimental results showed that in general, procedure $H1$ tends to yield higher makespan as we limit the buffer sizes even when the optimal makespan remains the same; although there is evidence that in some instances, the limitation on the buffer capacities can actually help the search procedure in finding a better result.

From the experimental results, we also determined that solving the FSPM/b problem using methods other than the FAM rule as employed in $H1$ is a difficult task (e.g., using procedure $H2$ together with procedure $Z1$). In many instances considered, procedure $H2$ fails to give a feasible solution. Accordingly, we conclude that in the context of procedure $H2$, procedure $Z1$ is not an effective procedure to generate the I and X matrices for a given solution vector S. This is due to the fact that procedure $Z1$ does not guarantee the feasibility of the resulting solution in the context of limited buffer capacities.

8.3 Future Research

From the results that we obtained in studying the FSPM and FSPM/b problems, we consider that the investigation can be further extended in some directions as follow.

- Given a solution vector S, we suggest to develop a different procedure other than procedures $Z1$, $Z2$, and FAM for constructing a complete schedule for the FSPM problem. One method that can be considered is to use a vector S to allocate the jobs at the bottle neck stage using the FAM rule. Then, a procedure is devised to schedule the jobs at the preceding and subsequent stages. The experiment is then performed to investigate its effectiveness as compared with procedure $Z1$ in solving the FSPM problem.

- For the FSPM problem, we recommend to conduct a further experiment in which at a given stage, the job processing time is proportional with the number of machines. This scenario represent the actual production line where more machines are allocated to stages that require longer time to process a type of job.

- In the context of FSPM/b problem with procedure $H2$, it is of interest to devise a procedure (other than procedure $Z1$, since we have determined that it is not an effective procedure in this context) that guarantees the feasibility of the solution generated. Then, together with procedure $H2$, its performance can be investigated in solving the FSPM/b problem (e.g. the results can be compared with the results obtained using procedure $H1$) to determine its effectiveness.

- In terms of the experiment with FSPM/b problem, we recommend to perform

another experiment using a similar scenario as that introduced by Wittrock [66], i.e., an experiment with a fix production line as defined by the number of stages (L), the number of machines at each stage (m_ℓ), and the buffer capacities (b_ℓ). The objective of the experiment is to study the impact of changing the number of jobs (N) and the job processing times ($p_{j\ell}$) on the makespan.

- In the context of FSPM/b problem, we used the set of lower bounds developed for FSPM problem in order to measure the solution quality. The experimental results show that this set of lower bounds are relatively strong for data set of type II, but we do not its strength in other instances. For future research, we recommend to develop the set of lower bounds which are specific for the FSPM/b problem.

- In the study that we have performed, we considered minimizing the makespan of the problem for given buffer capacities (either with unlimited or limited buffer capacities). For future research, we suggest to consider minimizing multiple objective functions such as minimizing both the makespan and the total number of buffer spaces utilized, or minimizing the makespan and at the same time, we consider how to allocate given buffer spaces among the buffer areas.

- Aside from considering minimizing makespan as the objective function, it is

also of an interest to study the effectiveness of the vector representation in solving the FSPM and FSPM/b problem with different objective functions, such as minimizing the maximum lateness, minimizing the total tardiness, etc.

Appendix A

Procedure to generate data set of type I

To construct a problem instance with known optimal makespan, we must first identify the number of stages L, the number of jobs N, and the number of machines at each stage. We assume that the number of machines at each stage is identical and we refer to it as m.

The next step is to construct the I and X matrices corresponding to the optimal schedule. In constructing these matrices, we make the following two assumptions.

1. In the optimal solution, the same collection of jobs are assigned to the same machine at every stage, i.e., all columns of the I matrix are identical.

2. The collection of jobs assigned to each machine are processed in the same order at all stages, i.e., all columns of the X matrix are identical.

To determine the collection of jobs assigned to each machine, we consider the following two cases.

- If N is divisible by m, we randomly allocate $\frac{N}{m}$ jobs to each machine.
- If N is not divisible by m, we randomly allocate $\lfloor \frac{N}{m} \rfloor$ jobs to each machine. Then, the remaining $N(\mathrm{mod}\, m)$ jobs are randomly allocated to $N(\mathrm{mod}\, m)$ machines (one job to each of these machines).

As a result, we know the entries in both the I and X matrices for the optimal schedule.

We now determine the job processing times in such a manner as to guarantee that the schedule is indeed optimal. We generate the job processing time $p_{j\ell}$ randomly according to the following two steps. First, we generate the processing time of jobs at the last stage L. Next, we generate the processing time of jobs at stages $L-1$ through 1, in that order. The procedures implemented in these two steps are described below. (Notice that in this discussion, $J_{i\ell}$ refers to the set of jobs processed on machine i at stage ℓ, for $i = 1, ..., m_\ell$ and $\ell = 1, ..., L$).

Procedure 1: Generating the job processing times at stage L

1. Let $T = \lfloor A + Bx \rfloor$, where $A = 50$, $B = 20$, and x is a random number from a uniform distribution between 0 and 1, i.e., $x \sim U[0, 1]$. Set the processing time of every job that is processed first on a machine equals to T, i.e.,

$$p_{jL} \quad = \quad T \quad \text{, for } \{j \in \{1, ..., N\} : X_{jL} = 1\}.$$

2. Generate the processing time of jobs processed on machine i, for $i = 1, ..., m$, as follow. Starting from job j with $X_{jL} = 2$ until job j with $X_{jL} = |J_{iL}|$, we determine the job processing time using the following formula:

$$p_{jL} \quad = \quad \lfloor p_{gL} + 1 + Bx \rfloor \quad \text{, for } \{j \in \{1, ..., N\} : 2 \leq X_{jL} \leq |J_{iL}|\}.$$

where g is the job processed immediately before job j on the same machine. (B and x are defined similarly as in step 1).

3. In this step, we modify (increase) the processing time of the last job on each machine (except on the machine that already has the longest total processing time). The modification is performed in such a manner that the total processing time of all job allocated to each machine are identical among all machines. To this end, let

$$C \quad = \quad \max_{i \in \{1,...,m\}} \left\{ \sum_{j \in J_{i\ell}} p_{jL} \right\}.$$

Let $j^*_{(i)}$ be the index of the last job that is processed on machine i, for $i = 1, ...m$. Then, we modify the processing time of this job at the last stage as follow.

$$p_{j^*_{(i)},L} \quad \leftarrow \quad p_{j^*_{(i)},L} + C - \sum_{j \in J_{i\ell}} p_{jL} \quad \text{, for all } i$$

It follows that after the modification, we have $\sum_{j \in J_{i\ell}} p_{jL} = C$, for all i.

Procedure 2: Generating the job processing times at stage $\ell < L$

Upon generating the job processing time at stage L, we recursively generate the job processing time at stage $\ell < L$. Starting from stage $\ell = L-1$ until stage $\ell = 1$, we perform the following steps.

4. Set the processing time of every job that is processed first on a machine as a fraction of its corresponding processing time at stage $\ell + 1$, i.e.,

$$p_{j\ell} \quad = \quad \lfloor \alpha \times p_{j,\ell+1} \rfloor \quad \text{, for all } \{j \in \{1, ..., N\} : X_{j\ell} = 1\}, (0 < \alpha < 1).$$

   In this experiment, we define $\alpha = 0.8$.

5. Set the processing time of job j with $X_{j\ell} > 1$ as follow.

$$p_{j\ell} \quad = \quad p_{g,\ell+1} \quad \text{, for all } \{j \in \{1, ..., N\} : X_{j\ell} > 1\}$$

   where g is the job processed immediately before job j on the same machine.

From the above procedures, we observe the following.

- At a given stage, the processing time of all jobs which are processed first on a machine are identical, and this value is smaller than the processing time of the remaining jobs at that stage. Therefore, the idle time of each machine at stage ℓ ($\ell > 1$) before it starts processing the first job is minimized.

- A job can always be processed immediately at stage $\ell + 1$ upon completion at stage ℓ. Consequently, there is no machine idle time at each stage. In addition, there is no buffer required between stages.

The following proposition shows that the schedule generated using the above procedure is optimal.

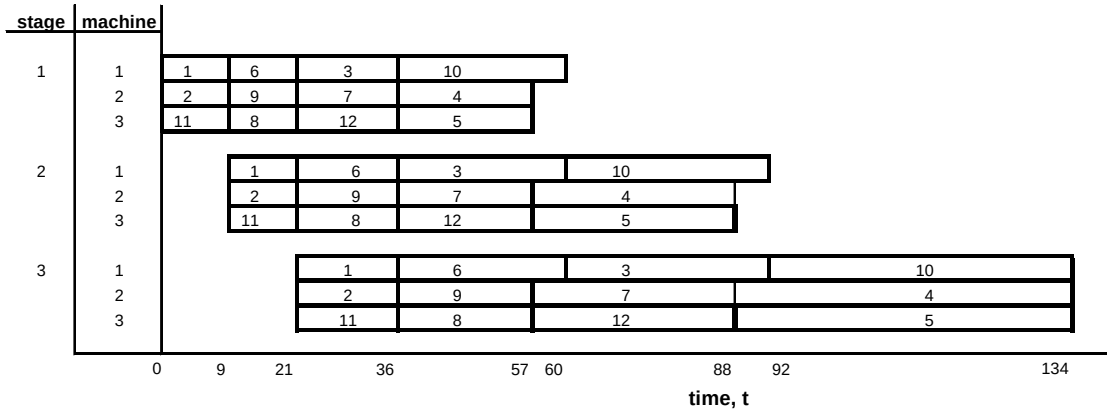
Proposition 3 *The schedule I, X given above is an optimal schedule for the corresponding instance.*

Proof The makespan $C_{\max}$ of the schedule can be obtained as follow.

$$C_{\max} = C + \sum_{\ell=1}^{L-1} \min_{j} \{p_{j\ell}\}$$

Calculating the lower bound of the instance using the method described in section 5.2, we determine that for $\ell = L$, $T_\ell = C$ and $\alpha_\ell = \sum_{\ell=1}^{L-1} \min_{j} \{p_{j\ell}\}$. Therefore, $LB_2 = C_{\max}$. This shows that the schedule generated using the above procedure is indeed optimal.□

In Figure A.1, we present a sample instance and its corresponding optimal schedule generated using the above procedure.



Matrix representation of the optimal schedule:

I =

job:	stage 1	2	3
1	1	1	1
2	2	2	2
3	1	1	1
4	2	2	2
5	3	3	3
6	1	1	1
7	2	2	2
8	3	3	3
9	2	2	2
10	1	1	1
11	3	3	3
12	3	3	3

X =

job:	stage 1	2	3
1	1	1	1
2	1	1	1
3	3	3	3
4	4	4	4
5	4	4	4
6	2	2	2
7	3	3	3
8	2	2	2
9	2	2	2
10	4	4	4
11	1	1	1
12	3	3	3

Processing time:

p =

job:	stage 1	2	3
1	9	12	15
2	9	12	15
3	15	24	32
4	21	30	47
5	21	31	46
6	12	15	24
7	15	21	30
8	12	15	21
9	12	15	21
10	24	32	42
11	9	12	15
12	15	21	31

Completion time:

C =

job:	stage 1	2	3
1	9	21	36
2	9	21	36
3	36	60	92
4	57	87	134
5	57	88	134
6	21	36	60
7	36	57	87
8	21	36	57
9	21	36	57
10	60	92	134
11	9	21	36
12	36	57	88

Figure A.1: A sample instance generated with known optimal makespan.

Appendix B

Miscellaneous results

In this appendix, we present the results that support the experiment that we made.

In Tables B.1 and B.2, we present the number of iterations obtained upon running the program for 30 seconds of CPU time using data sets of type I and type II, respectively. These tables correspond with the experimental results shown in sections 5.3.1 to 5.3.4. With regard to the local improvement procedure, the number of iterations refers to the number of starting points generated. In the genetic algorithm, the number of iterations refers to the number of population generations.

In Table B.3, we present the similar results that correspond with the experimental results shown in section 5.3.5, i.e., the number of iterations performed upon running the program until the stopping criterion is met.

Table B.1: The average number of iterations within 30 seconds of CPU time for various algorithms (data set of type I).

N x L x m_L	LIM	LI-Z1-F	LI-Z1-G	LI-Z1-L	LI-Z2-F	LI-Z2-G	LI-Z2-L	TS-Z1	TS-Z2	GA-Z1	GA-Z2
20 x 2 x 2	1.0	1948.0	104.0	2321.2	2108.2	115.8	2562.8	3603.4	3901.4	1237.2	1434.2
20 x 2 x 4	1.4	1988.4	129.0	2283.6	2143.0	139.8	2432.2	3009.2	3210.2	993.0	1128.0
20 x 2 x 6	3.0	1757.8	123.2	2283.4	1877.0	136.4	2442.0	2488.4	2661.2	874.0	925.8
20 x 4 x 2	1.4	1008.6	39.0	1368.4	1090.8	41.6	1502.8	2499.0	2760.2	711.8	849.2
20 x 4 x 4	2.0	1277.0	63.0	1384.6	1376.0	73.6	1501.0	1935.0	2065.0	560.8	620.4
20 x 4 x 6	2.6	1263.2	75.4	1410.2	1355.6	83.8	1499.2	1468.0	1547.0	438.8	471.8
20 x 6 x 2	1.2	606.0	20.6	917.0	680.8	25.2	1035.4	1940.0	2176.6	512.0	586.6
20 x 6 x 4	1.6	854.4	42.4	961.2	948.4	48.8	1053.0	1412.6	1530.6	373.6	412.4
20 x 6 x 6	2.4	887.0	58.2	1026.6	969.0	59.6	1083.6	1018.8	1111.6	303.4	328.6
20 x 8 x 2	2.8	445.0	16.6	707.2	498.8	18.6	800.0	1556.6	1783.2	399.2	456.4
20 x 8 x 4	2.6	652.0	34.0	751.4	738.8	39.8	822.2	1070.8	1190.4	288.0	336.2
20 x 8 x 6	2.6	702.6	48.8	824.2	775.0	52.0	881.6	801.0	868.4	230.4	254.6
50 x 2 x 2	1.0	297.4	5.4	561.0	322.6	6.0	594.2	662.8	793.6	509.6	548.2
50 x 2 x 4	1.0	387.2	9.4	464.0	418.6	10.4	500.4	557.6	562.6	396.2	410.6
50 x 2 x 6	1.0	369.4	10.0	424.0	397.8	9.0	454.8	494.8	487.4	343.8	346.8
50 x 4 x 2	1.4	96.6	2.6	210.8	106.4	2.2	231.6	332.4	360.8	260.6	281.4
50 x 4 x 4	1.0	163.8	3.8	187.0	172.4	3.6	194.8	236.2	260.0	201.4	209.0
50 x 4 x 6	2.2	184.8	4.4	179.2	190.8	3.2	190.6	231.0	221.2	169.8	177.8
50 x 6 x 2	1.8	48.8	2.0	126.4	55.4	2.2	129.8	198.2	205.8	175.4	192.6
50 x 6 x 4	2.0	87.0	2.2	104.2	98.6	2.4	113.8	153.2	171.0	129.6	137.4
50 x 6 x 6	2.4	102.0	2.8	102.0	116.0	2.6	111.8	138.6	155.6	116.0	119.2
50 x 8 x 2	1.8	28.2	2.0	80.2	34.8	2.0	90.8	123.6	141.6	126.4	133.2
50 x 8 x 4	2.4	58.0	2.2	75.2	60.2	2.2	80.2	112.6	114.0	99.8	105.2
50 x 8 x 6	1.8	65.8	2.4	76.2	76.4	2.4	80.4	113.4	101.6	83.6	89.0
100 x 2 x 2	1.0	48.2	2.0	127.8	51.6	2.0	118.2	290.2	274.6	190.6	209.4
100 x 2 x 4	1.0	85.2	2.2	101.0	87.2	2.2	101.2	225.6	257.8	155.6	150.4
100 x 2 x 6	2.4	83.6	2.0	95.2	89.4	2.0	99.4	202.6	223.4	148.2	132.8
100 x 4 x 2	1.2	16.0	2.0	48.2	16.0	2.0	53.2	96.8	145.0	93.0	95.6
100 x 4 x 4	2.4	28.2	2.0	35.2	30.2	2.0	37.8	69.0	103.8	55.6	66.8
100 x 4 x 6	1.2	30.6	2.0	30.6	31.6	2.0	31.8	74.2	89.0	65.2	65.0
100 x 6 x 2	2.2	8.0	2.0	31.2	9.4	2.0	33.0	103.2	86.2	62.0	58.4
100 x 6 x 4	2.0	17.8	2.0	18.0	18.2	2.0	22.6	61.8	67.4	45.2	47.2
100 x 6 x 6	1.8	20.6	2.0	18.0	18.4	2.0	21.2	62.0	57.8	41.2	43.0
100 x 8 x 2	1.4	6.0	2.0	18.6	4.6	2.0	20.6	79.8	93.4	43.8	41.4
100 x 8 x 4	1.2	8.8	2.0	12.0	12.2	2.0	13.8	50.4	54.6	33.8	35.6
100 x 8 x 6	1.8	12.0	2.0	13.6	13.0	2.0	12.8	46.6	47.2	27.0	30.4
150 x 2 x 2	1.0	16.0	2.0	65.8	17.4	2.0	69.6	167.2	181.0	112.4	109.6
150 x 2 x 4	1.0	32.2	2.0	42.2	32.2	2.0	47.6	139.4	141.2	91.6	90.4
150 x 2 x 6	1.0	30.4	2.0	38.8	34.4	2.0	43.8	120.2	121.0	71.2	71.8
150 x 4 x 2	1.0	5.4	2.0	20.6	6.2	2.0	22.0	90.2	67.6	47.4	51.2
150 x 4 x 4	1.0	11.8	2.0	15.2	12.6	2.0	14.2	56.2	57.2	38.0	38.2
150 x 4 x 6	1.2	12.6	2.0	13.4	12.8	2.0	15.2	58.8	54.2	28.2	29.8
150 x 6 x 2	1.4	4.0	2.0	12.0	4.0	2.0	12.6	64.6	60.2	28.0	31.8
150 x 6 x 4	1.2	7.4	2.0	9.4	8.4	2.0	8.2	56.8	46.8	19.0	21.2
150 x 6 x 6	1.0	7.2	2.0	7.2	9.0	2.0	8.8	40.8	34.4	18.6	21.0
150 x 8 x 2	1.2	3.4	2.0	9.8	2.8	2.0	9.4	40.0	60.4	23.4	23.0
150 x 8 x 4	1.0	4.4	2.0	5.8	6.4	2.0	6.4	47.6	38.0	17.4	13.2
150 x 8 x 6	1.2	5.4	2.0	5.0	7.0	2.0	5.6	39.4	32.6	12.8	14.6
Average	1.6	328.8	17.7	408.8	357.2	19.4	442.0	592.5	641.2	229.2	251.0

Table B.2: The average number of iterations within 30 seconds of CPU time for various algorithms (data set of type II).

N x L x m_L	LIM	LI-Z1-F	LI-Z1-G	LI-Z1-L	LI-Z2-F	LI-Z2-G	LI-Z2-L	TS-Z1	TS-Z2	GA-Z1	GA-Z2
20 x 2 x 2	2.6	1313.2	96.8	2096.2	1412.4	105.8	2252.0	3329.0	3514.8	1016.0	1128.8
20 x 2 x 4	4.6	1301.4	70.4	1617.6	1393.6	76.6	1741.0	3354.0	3590.6	969.6	1101.8
20 x 2 x 6	7.0	1271.0	75.0	1541.4	1353.6	83.0	1668.0	2817.0	2985.0	868.4	965.4
20 x 4 x 2	1.4	962.0	42.4	1200.2	1112.8	43.6	1205.2	2146.8	2439.0	576.4	645.4
20 x 4 x 4	1.6	841.8	46.8	1071.6	904.6	47.2	1143.2	1971.4	2120.0	522.4	570.8
20 x 4 x 6	5.6	785.0	44.6	936.0	829.8	48.0	999.4	1740.8	1845.0	474.2	510.6
20 x 6 x 2	2.0	767.2	31.4	851.2	927.0	33.4	960.2	1655.0	1882.2	424.6	472.0
20 x 6 x 4	2.2	623.8	34.0	752.8	712.0	34.4	812.4	1445.4	1577.2	363.2	396.2
20 x 6 x 6	1.8	531.4	41.2	709.6	573.8	44.0	764.2	1123.8	1213.6	316.0	337.2
20 x 8 x 2	3.0	653.6	29.6	730.6	775.0	33.8	859.6	1395.4	1540.8	332.0	381.6
20 x 8 x 4	3.0	505.0	27.2	622.6	548.2	28.4	700.0	1088.0	1189.4	259.0	287.6
20 x 8 x 6	5.2	440.2	26.0	548.8	489.4	28.6	595.8	961.2	1034.6	242.0	260.2
50 x 2 x 2	2.0	201.4	9.2	474.0	225.8	11.2	508.6	583.8	630.6	415.2	440.0
50 x 2 x 4	3.8	201.4	6.4	386.8	211.4	7.8	400.6	528.4	552.4	336.6	349.6
50 x 2 x 6	2.8	208.8	7.2	291.0	224.8	6.4	308.4	499.8	508.0	317.4	321.6
50 x 4 x 2	2.4	115.0	3.4	206.6	130.8	3.0	173.6	289.2	299.6	201.2	213.0
50 x 4 x 4	2.6	101.0	2.8	143.2	109.2	2.6	144.4	253.0	240.8	168.4	181.6
50 x 4 x 6	3.4	101.8	3.6	123.6	112.4	2.8	124.6	222.0	237.6	158.0	165.2
50 x 6 x 2	1.8	81.4	3.0	117.8	93.4	2.2	125.4	223.0	213.4	147.8	156.6
50 x 6 x 4	2.0	77.2	2.8	92.4	87.4	2.0	100.6	160.2	174.2	114.6	119.8
50 x 6 x 6	4.6	71.2	2.6	80.4	83.6	2.6	89.8	143.6	165.6	104.4	113.2
50 x 8 x 2	1.6	79.8	2.2	86.6	96.6	2.8	102.6	134.0	166.0	104.4	119.2
50 x 8 x 4	2.8	70.4	2.6	72.4	83.2	2.2	86.6	116.0	137.0	86.6	91.8
50 x 8 x 6	4.4	59.4	2.6	69.8	69.8	2.4	84.2	89.8	122.6	76.0	81.0
100 x 2 x 2	3.0	37.6	2.0	110.6	38.6	2.2	120.4	292.0	272.4	167.2	180.2
100 x 2 x 4	3.2	32.0	2.0	90.2	38.4	2.0	97.4	217.4	248.0	132.4	130.0
100 x 2 x 6	5.8	45.4	2.0	69.4	48.2	2.0	75.4	213.0	212.6	132.8	122.4
100 x 4 x 2	1.6	20.0	2.0	47.8	22.6	2.0	26.0	129.6	114.0	75.4	81.8
100 x 4 x 4	1.2	16.8	2.0	35.6	20.4	2.0	23.6	95.2	97.4	50.0	60.6
100 x 4 x 6	1.8	22.6	2.0	23.8	18.4	2.0	23.6	84.0	89.6	59.8	61.8
100 x 6 x 2	2.4	19.0	2.0	22.0	20.8	2.0	21.6	85.2	78.8	55.2	54.0
100 x 6 x 4	2.2	15.2	2.0	20.6	19.4	2.0	19.6	63.0	61.8	40.8	45.2
100 x 6 x 6	2.0	17.2	2.0	13.8	15.6	2.0	19.8	59.6	59.4	38.6	40.0
100 x 8 x 2	1.4	17.4	2.0	16.8	20.0	2.0	21.0	55.2	65.6	41.8	41.6
100 x 8 x 4	2.2	14.2	2.0	13.2	17.2	2.0	19.0	53.8	39.2	32.4	33.8
100 x 8 x 6	4.0	11.2	2.0	13.6	15.0	2.0	14.2	38.6	43.6	26.2	31.2
150 x 2 x 2	1.2	11.0	2.0	66.0	9.0	2.0	67.8	155.8	153.2	82.6	85.2
150 x 2 x 4	2.6	14.8	2.0	37.6	15.0	2.0	38.8	144.4	153.2	77.6	83.4
150 x 2 x 6	2.8	12.2	2.0	28.2	14.0	2.0	31.8	117.2	131.2	65.6	69.6
150 x 4 x 2	1.4	6.4	2.0	17.4	7.4	2.0	10.4	80.4	74.4	40.4	46.4
150 x 4 x 4	1.6	7.8	2.0	8.4	6.6	2.0	7.8	64.8	59.8	36.2	37.8
150 x 4 x 6	1.6	7.8	2.0	11.0	6.0	2.0	8.0	51.2	73.6	28.0	28.6
150 x 6 x 2	1.0	6.6	2.0	6.4	7.2	2.0	7.4	47.8	45.2	26.2	31.8
150 x 6 x 4	1.6	5.0	2.0	7.0	7.2	2.0	7.0	37.2	41.4	18.6	21.8
150 x 6 x 6	1.2	5.8	2.0	6.2	6.2	2.0	7.0	37.6	37.0	19.0	21.0
150 x 8 x 2	1.4	6.6	2.0	6.0	7.8	2.0	6.8	35.0	34.4	22.4	24.4
150 x 8 x 4	1.0	6.4	2.0	5.4	8.2	2.0	6.2	31.0	24.0	17.6	13.6
150 x 8 x 6	1.2	6.8	2.0	4.4	6.8	2.0	6.2	22.4	22.8	12.4	14.4
Average	2.6	244.4	13.8	323.0	269.9	14.6	346.6	593.4	637.8	206.1	224.4

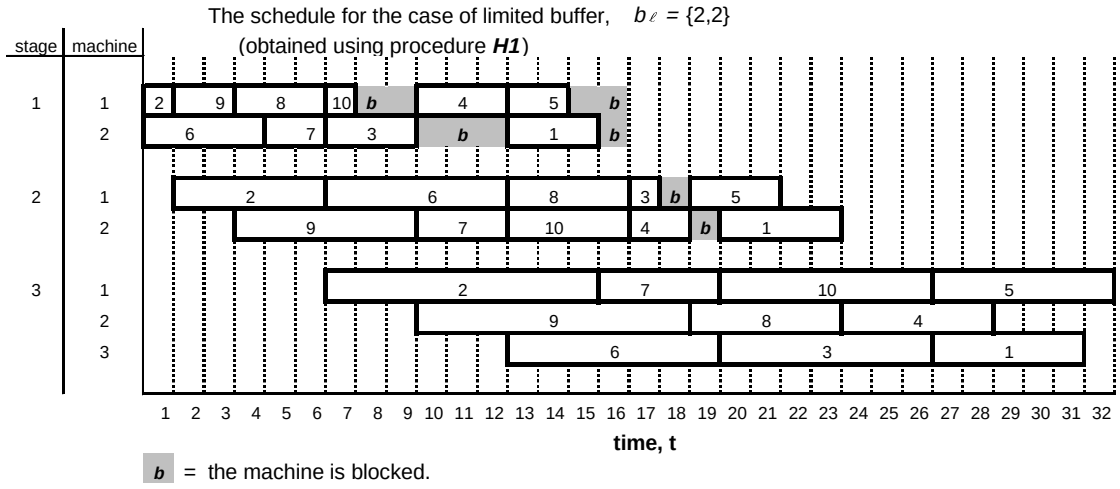
Table B.3: The average number of iterations for algorithms TS-Z1 and GA-Z1 upon running the program until the stopping criterion is met (data sets of type I and II).

	Data set of type I		Data set of type II	
N x L x m_L	TS-Z1	GA-Z1	TS-Z1	GA-Z1
20 x 2 x 2	176.4	275.0	114.2	113.8
20 x 2 x 4	201.0	173.0	212.4	176.6
20 x 2 x 6	165.6	229.8	198.8	181.6
20 x 4 x 2	223.6	241.2	183.4	185.2
20 x 4 x 4	242.8	198.0	259.4	210.0
20 x 4 x 6	166.4	147.6	203.4	191.0
20 x 6 x 2	199.4	220.6	200.8	223.0
20 x 6 x 4	240.6	233.6	197.0	232.0
20 x 6 x 6	181.4	157.8	154.8	121.4
20 x 8 x 2	222.8	220.6	151.2	244.6
20 x 8 x 4	254.0	239.6	231.2	226.4
20 x 8 x 6	155.8	148.4	185.6	210.2
50 x 2 x 2	285.6	250.2	143.4	190.2
50 x 2 x 4	344.6	244.6	198.0	184.2
50 x 2 x 6	250.6	293.8	279.4	298.8
50 x 4 x 2	372.2	373.0	282.6	186.8
50 x 4 x 4	404.2	249.8	289.6	255.0
50 x 4 x 6	206.0	240.6	286.2	359.8
50 x 6 x 2	360.0	324.8	165.6	252.0
50 x 6 x 4	283.6	324.8	248.6	350.8
50 x 6 x 6	339.6	336.8	339.4	309.6
50 x 8 x 2	439.4	471.8	381.4	322.0
50 x 8 x 4	308.4	409.4	291.8	292.6
50 x 8 x 6	369.2	205.6	385.4	321.0
100 x 2 x 2	506.4	310.6	142.0	141.0
100 x 2 x 4	350.2	224.0	230.2	174.4
100 x 2 x 6	328.2	281.6	311.8	341.4
100 x 4 x 2	371.6	384.8	299.4	218.0
100 x 4 x 4	499.6	364.8	316.2	380.4
100 x 4 x 6	387.2	346.0	296.4	363.8
100 x 6 x 2	412.0	550.8	309.2	295.2
100 x 6 x 4	485.4	318.4	359.2	289.0
100 x 6 x 6	354.6	342.4	437.2	304.2
100 x 8 x 2	511.4	318.6	218.2	317.8
100 x 8 x 4	460.6	338.2	430.6	246.8
100 x 8 x 6	380.6	346.6	378.4	233.6
150 x 2 x 2	340.4	378.4	171.0	133.0
150 x 2 x 4	391.8	250.4	170.0	155.4
150 x 2 x 6	331.0	314.0	317.0	207.6
150 x 4 x 2	562.8	572.0	220.4	180.4
150 x 4 x 4	555.6	408.6	305.8	252.2
150 x 4 x 6	418.6	435.2	299.0	219.4
150 x 6 x 2	525.6	489.6	344.8	335.4
150 x 6 x 4	467.2	462.2	506.2	402.6
150 x 6 x 6	334.4	341.4	533.0	277.6
150 x 8 x 2	656.2	430.8	438.4	342.0
150 x 8 x 4	571.4	432.2	535.0	235.6
150 x 8 x 6	437.0	422.2	497.6	303.4
Average	354.9	318.2	284.4	249.8

Appendix C

A further example of procedure $H1$

In this appendix, we present another small example to further illustrate procedure $H1$. We consider an instance of 10 jobs with the job processing times as shown in Figure C.1. There are three stages with the number of machines $m_1 = 2$, $m_2 = 2$, and $m_3 = 3$ with buffer capacities $b_1 = b_2 = 2$. Given a solution vector $S = \{2, 6, 9, 8, 7, 10, 3, 4, 1, 5\}$ we construct a complete schedule using procedure $H1$. The step-by-step of the procedure is shown in Table C.1. The corresponding matrix representation is presented in Figure C.1.



Vector representation : ***S =*** {2,6,9,8,7,10,3,4,1,5}

Matrix representation:

I =

job:	stage 1	2	3
1	2	2	3
2	1	1	1
3	2	1	3
4	1	2	2
5	1	1	1
6	2	1	3
7	2	2	1
8	1	1	2
9	1	2	2
10	1	2	1

X =

job:	stage 1	2	3
1	4	5	3
2	1	1	1
3	3	4	2
4	5	4	3
5	6	5	4
6	1	2	1
7	2	2	2
8	3	3	2
9	2	1	1
10	4	3	3

Processing time

P =

job:	stage 1	2	3
1	3	4	5
2	1	5	9
3	3	1	7
4	3	2	5
5	2	3	6
6	4	6	7
7	2	3	4
8	3	4	5
9	2	6	9
10	1	4	7

Completion time

C =

job:	stage 1	2	3
1	15	23	31
2	1	6	15
3	9	17	26
4	12	18	28
5	14	21	32
6	4	12	19
7	6	12	19
8	6	16	23
9	3	9	18
10	7	16	26

Figure C.1: Another example of the implementation of procedure $H1$.

Table C.1: The step-by-step of the implementation of procedure $H1$.

Time	Flow of the jobs on machines or buffers *)											Event list	Event	Completed jobs
	ℓ = 1		b_1		ℓ = 2		b_2		ℓ = 3					
t	m_1	m_2	1	2	m_1	m_2	1	2	m_1	m_2	m_3	*E*	time	*R*
0	2											$\{C_{21}\}$	{1}	
	2	6										$\{C_{21},C_{61}\}$	{1,4}	
1		6			2							$\{C_{61},C_{22}\}$	{4,6}	
	9	6			2							$\{C_{91},C_{61},C_{22}\}$	{3,4,6}	
3		6			2	9						$\{C_{61},C_{22},C_{92}\}$	{4,6,9}	
	8	6			2	9						$\{C_{61},C_{22},C_{81},C_{92}\}$	{4,6,6,9}	
4	8		6		2	9						$\{C_{22},C_{81},C_{92}\}$	{6,6,9}	
	8	7	6		2	9						$\{C_{22},C_{81},C_{71},C_{92}\}$	{6,6,6,9}	
6	8	7	6			9			2			$\{C_{81},C_{71},C_{92},C_{23}\}$	{6,6,9,15}	
	8	7			6	9			2			$\{C_{81},C_{71},C_{92},C_{62},C_{23}\}$	{6,6,9,12,15}	
6		7	8		6	9			2			$\{C_{71},C_{92},C_{62},C_{23}\}$	{6,9,12,15}	
	10	7	8		6	9			2			$\{C_{71},C_{10,1},C_{92},C_{62},C_{23}\}$	{6,7,9,12,15}	
6	10		8	7	6	9			2			$\{C_{10,1},C_{92},C_{62},C_{23}\}$	{7,9,12,15}	
	10	3	8	7	6	9			2			$\{C_{10,1},C_{92},C_{31},C_{62},C_{23}\}$	{7,9,9,12,15}	
7	10**b**	3	8	7	6	9			2			$\{C_{92},C_{31},C_{62},C_{23}\}$	{9,9,12,15}	
9	10**b**	3	8	7	6				2	9		$\{C_{31},C_{62},C_{23},C_{93}\}$	{9,12,15,18}	
	10**b**	3	8		6	7			2	9		$\{C_{31},C_{62},C_{72},C_{23},C_{93}\}$	{9,12,12,15,18}	
		3	8	10	6	7			2	9		$\{C_{31},C_{62},C_{72},C_{23},C_{93}\}$	{9,12,12,15,18}	
	4	3	8	10	6	7			2	9		$\{C_{31},C_{62},C_{72},C_{41},C_{23},C_{93}\}$	{9,12,12,12,15,18}	
9	4	3**b**	8	10	6	7			2	9		$\{C_{62},C_{72},C_{41},C_{23},C_{93}\}$	{12,12,12,15,18}	
12	4	3**b**	8	10		7			2	9	6	$\{C_{72},C_{41},C_{23},C_{93},C_{63}\}$	{12,12,15,18,19}	
	4	3**b**		10	8	7			2	9	6	$\{C_{72},C_{41},C_{23},C_{82},C_{93},C_{63}\}$	{12,12,15,16,18,19}	
	4		3	10	8	7			2	9	6	$\{C_{72},C_{41},C_{23},C_{82},C_{93},C_{63}\}$	{12,12,15,16,18,19}	
	4	1	3	10	8	7			2	9	6	$\{C_{72},C_{41},C_{23},C_{11},C_{82},C_{93},C_{63}\}$	{12,12,15,15,16,18,19}	
12	4	1	3	10	8		7		2	9	6	$\{C_{41},C_{23},C_{11},C_{82},C_{93},C_{63}\}$	{12,15,15,16,18,19}	
	4	1	3		8	10	7		2	9	6	$\{C_{41},C_{23},C_{11},C_{82},C_{10,2},C_{93},C_{63}\}$	{12,15,15,16,16,18,19}	
12		1	3	4	8	10	7		2	9	6	$\{C_{23},C_{11},C_{82},C_{10,2},C_{93},C_{63}\}$	{15,15,16,16,18,19}	
	5	1	3	4	8	10	7		2	9	6	$\{C_{51},C_{23},C_{11},C_{82},C_{10,2},C_{93},C_{63}\}$	{14,15,15,16,16,18,19}	
14	5**b**	1	3	4	8	10	7		2	9	6	$\{C_{23},C_{11},C_{82},C_{10,2},C_{93},C_{63}\}$	{15,15,16,16,18,19}	
15	5**b**	1	3	4	8	10	7			9	6	$\{C_{11},C_{82},C_{10,2},C_{93},C_{63}\}$	{15,16,16,18,19}	{2}
	5**b**	1	3	4	8	10			7	9	6	$\{C_{11},C_{82},C_{10,2},C_{93},C_{73},C_{63}\}$	{15,16,16,18,19,19}	{2}
15	5**b**	1**b**	3	4	8	10			7	9	6	$\{C_{82},C_{10,2},C_{93},C_{73},C_{63}\}$	{16,16,18,19,19}	{2}
16	5**b**	1**b**	3	4		10	8		7	9	6	$\{C_{10,2},C_{93},C_{73},C_{63}\}$	{16,18,19,19}	{2}
	5**b**	1**b**		4	3	10	8		7	9	6	$\{C_{10,2},C_{32},C_{93},C_{73},C_{63}\}$	{16,17,18,19,19}	{2}
		1**b**	5	4	3	10	8		7	9	6	$\{C_{10,2},C_{32},C_{93},C_{73},C_{63}\}$	{16,17,18,19,19}	{2}
16		1**b**	5	4	3		8	10	7	9	6	$\{C_{32},C_{93},C_{73},C_{63}\}$	{17,18,19,19}	{2}
		1**b**	5		3	4	8	10	7	9	6	$\{C_{32},C_{93},C_{42},C_{73},C_{63}\}$	{17,18,18,19,19}	{2}
			5	1	3	4	8	10	7	9	6	$\{C_{32},C_{93},C_{42},C_{73},C_{63}\}$	{17,18,18,19,19}	{2}
17			5	1	3**b**	4	8	10	7	9	6	$\{C_{93},C_{42},C_{73},C_{63}\}$	{18,18,19,19}	{2}
18			5	1	3**b**	4	8	10	7		6	$\{C_{42},C_{73},C_{63}\}$	{18,19,19}	{2,9}
			5	1	3**b**	4		10	7	8	6	$\{C_{42},C_{73},C_{63},C_{83}\}$	{18,19,19,23}	{2,9}
			5	1		4	3	10	7	8	6	$\{C_{42},C_{73},C_{63},C_{83}\}$	{18,19,19,23}	{2,9}
				1	5	4	3	10	7	8	6	$\{C_{42},C_{73},C_{63},C_{52},C_{83}\}$	{18,19,19,21,23}	{2,9}

Table C.1: The step-by-step of the implementation of procedure $H1$ (Continued).

Time	Flow of the jobs on machines or buffers *)											Event list	Event	Completed jobs
	ℓ = 1		b_1		ℓ = 2		b_2		ℓ = 3					
t	m_1	m_2	1	2	m_1	m_2	1	2	m_1	m_2	m_3	E	time	R
18			1		5	**4b**	3	10	7	8	6	$\{C_{73},C_{63},C_{52},C_{83}\}$	{19,19,21,23}	{2,9}
19			1		5	**4b**	3	10		8	6	$\{C_{63},C_{52},C_{83}\}$	{19,21,23}	{2,9,7}
			1		5	**4b**	3		10	8	6	$\{C_{63},C_{52},C_{83},C_{10,3}\}$	{19,21,23,26}	{2,9,7}
			1		5		3	4	10	8	6	$\{C_{63},C_{52},C_{83},C_{10,3}\}$	{19,21,23,26}	{2,9,7}
					5	1	3	4	10	8	6	$\{C_{63},C_{52},C_{83},C_{12},C_{10,3}\}$	{19,21,23,23,26}	{2,9,7}
19					5	1	3	4	10	8		$\{C_{52},C_{83},C_{12},C_{10,3}\}$	{21,23,23,26}	{2,9,7,6}
					5	1		4	10	8	3	$\{C_{52},C_{83},C_{12},C_{10,3},C_{33}\}$	{21,23,23,26,26}	{2,9,7,6}
21						1	5	4	10	8	3	$\{C_{83},C_{12},C_{10,3},C_{33}\}$	{23,23,26,26}	{2,9,7,6}
23						1	5	4	10		3	$\{C_{12},C_{10,3},C_{33}\}$	{23,26,26}	{2,9,7,6,8}
						1	5		10	4	3	$\{C_{12},C_{10,3},C_{33},C_{43}\}$	{23,26,26,28}	{2,9,7,6,8}
23							5	1	10	4	3	$\{C_{10,3},C_{33},C_{43}\}$	{26,26,28}	{2,9,7,6,8}
26							5	1		4	3	$\{C_{33},C_{43}\}$	{26,28}	{2,9,7,6,8,10}
								1	5	4	3	$\{C_{33},C_{43},C_{53}\}$	{26,28,32}	{2,9,7,6,8,10}
26								1	5	4		$\{C_{43},C_{53}\}$	{28,32}	{2,9,7,6,8,10,3}
									5	4	1	$\{C_{43},C_{13},C_{53}\}$	{28,31,32}	{2,9,7,6,8,10,3}
28									5		1	$\{C_{13},C_{53}\}$	{31,32}	{2,9,7,6,8,10,3,4}
31									5			$\{C_{53}\}$	{32}	{2,9,7,6,8,10,3,4,1}
32												{-}	{}	{2,9,7,6,8,10,3,4,1,5}

*) = index **b** following a job number means that the job is blocking the machine.

Bibliography

[1] Aartz, E. and Korst, J. (1989) "Simulated Annealing and Boltzman Machines", John Wiley & Sons Inc., New York.

[2] Armentano, Vinicius A. and Debora P. Ronconi (1999), "Tabu search for total tardiness minimization in flowshop scheduling problems", *Computers & Operation Research*, Vol. 26, pp. 219-235.

[3] Baker, Kenneth R. (1974) "Introduction to sequencing and scheduling", John Wiley & Sons, New York.

[4] Ben-Daya, M. and M. Al-Fawzan (1998), "A tabu search approach for the flow shop scheduling problem", *European Journal of Operational Research*, Vol. 109, pp. 88-95.

[5] Bianco, Lucio; Paolo Dell'olmo; and Stefano Giordani (1999), "Flow shop no-wait scheduling with sequence dependent setup times and release dates", *INFOR*, Vol. 37, pp. 3-19.

[6] Bjorndal M.H.; A. Capara; P.I. Cowling; F. Della Croce; H. Lourenco; F. Malucelli; A.J. Orman; D. Pisinger; C. Rego; and J.J. Salazar (1995), "Some thoughts on combinatorial optimization", *European Journal of Operational Research*, Vol. 83, pp. 253-270.

[7] Campbell, Herbert G.; Dudek, Richard A.; and Smith, Milton J. (1970) "A heuristic algorithm for the n job, m machine sequencing problem", *Management science*, Vol. 16, No. 10, pp. B630-B637.

[8] Carlier, Jacques (1987), "Scheduling jobs with release dates and tails on identical machine to minimize the makespan", *European Journal of Operational Research*, Vol. 29, pp. 297-306.

[9] Chen, Bo (1994), "Scheduling multiprocessor flow shops", in *Advances in Optimization and Approximation* (Du and Sun, Eds), pp. 1-8.

[10] Chen, Bo (1995), "Analysis of classes of heuristics for scheduling a two-stage flow shop with parallel machines at one stage", *Journal of the Operational Research Society*, Vol. 46, pp. 234-244.

[11] Chen, Bo; Glass, Celia A.; Potts, Chris N.; and Strusevich, Vitaly A. (1996), "A new heuristic for Three-Machine Flow Shop Scheduling", *Operations Research*, Vol. 44, No. 6, pp. 891-898.

[12] Chiang, Wen-Chyuan and Chiang, Chi (1998), "Intelligent local search strategies for solving facility layout problem with quadratic assignment problem formulation", *European Journal of Operational Research*, Vol. 106, pp. 457-488.

[13] Dannenbring, David G. (1977), "An evaluation of flowshop sequencing heuristics", *Management science*, Vol. 23, No. 11, pp. 1174-1182.

[14] Das, H.; Cummings, P.T.; and Van, Le, M.D. (1990), "Scheduling of serial multiproduct batch processes via simulated annealing", *Computers and Chemical Engineering*, Vol. 14, No. 12, pp. 1351-1362.

[15] David, S. Johnson; Cecilia, R. Aragon; Lyle, A. McGeoch; and Catherine Schevon (1989), "Optimization by simulated annealing: An experimental evaluation; Part I, Graph Partitioning", *Operation Research*, Vol. 37, No. 6, pp. 865-892.

[16] David, S. Johnson; Cecilia, R. Aragon; Lyle, A. McGeoch; and Catherine Schevon (1989), "Optimization by simulated annealing: An experimental evaluation; Part II, Graph Coloring and Number Partitioning", *Operation Research*, Vol. 39, No. 3, pp. 378-406.

[17] David, T. Connoly (1990), "An improved annealing scheme for the QAP", *European Journal of Operational Research*, Vol. 46, pp. 93-100.

[18] Ding, F.Y. and Kittichartphayak, D. (1994), "Heuristics for scheduling flexible flow lines", *Computers & Industrial Engineering*, Vol. 26, pp. 27-34.

[19] Elmaghraby, Salah E. and Kristin A. Thoney (1999), "The two machines stochastic flowshop problem with arbitrary processing time distributions", *IIE Transactions*, Vol. 31, pp. 467-477.

[20] Garey, M.R.; Johnson, D.S.; Sethi, Ravi (1976), "The complexity of flowshop and jobshop scheduling", *Mathematics of Operations Research*, Vol. 1, No. 2, pp. 117-129.

[21] Gen, Mitsuo and Cheng, Runwei (1997), "Genetic Algorithms & Engineering Design", John Wiley & Sons Inc., New York.

[22] Glover, Fred and Laguna, Manuel (1997), "Tabu Search", Kluwer Academic Publishers, Boston/Dordrecht/London.

[23] Gupta, N.D. Jatinder (1986), "Flowshop schedules with sequence dependent setup times", *Journal of the Operations Research Society of Japan*, Vol. 29, No. 3, pp. 206-219.

[24] Guinet, A.; M.M. Solomon; P.K. Kedia; and A. Dussauchoy (1996), "A computational study of heuristic for two-stage flexible flow shops", *International Journal of Production Research*, Vol. 34, pp. 1399-1415.

[25] Haouari, Mohammed and Rym, M'hallah (1997), "Heuristic algorithms for the two-stage hybrid flowshop problem", *Operations Research Letters*, Vol. 21, pp43-53.

[26] Hunsucker, J.L. and Shah, J.R. (1994), "Comparative performance analysis of priority rules in a constrained flow shop with multiple processors environment", *European Journal of Operational Research*, Vol. 72, pp. 102-114.

[27] Johnson, S.M. (1954), "Optimal two- and three-stage production schedules with setup times included", *Naval Research Logistics Quarterly*, Vol. 1, No. 1, pp. 61-68.

[28] Kellerer, Hans (1998), "Algorithms for multiprocessor scheduling with machine release times", *IIE Transactions*, Vol. 30, pp. 991-999.

[29] King, J.R. and Spachis, A.S. (1980), "Heuristic for flow-shop scheduling", *International Journal of Production Research*, Vol. 18, No. 3, pp. 345-357.

[30] Kuik, R. and Salomon, M. (1990), "Multi-level lot sizing problem: Evaluation of a simulated-annealing heuristic", *European Journal of Operational Research*, Vol. 45, pp. 25-37.

[31] Labetoulle, J.; Lawler, E.L.; Lenstra, J.K.; and Rinnoy Khan, A.H.G. (1984) "Preemptive scheduling of uniform machines subject to release dates", *Progress in Combinatorial Optimization*, Academic Press, Orlando, FL., pp. 245-261.

[32] Lee, Chung-Yee and George L. Vairaktarakis (1994), "Minimizing makespan in hybrid flowshops", *Operation Research Letters*, Vol. 16, pp. 149-158.

[33] Leisten, Rainer (1990), "Flowshop sequencing problems with limited buffer storage", *International Journal of Production Research*, Vol. 28, No. 11, pp. 2085-2100.

[34] Levner, E.M. (1969) "Optimal planning of parts' machining on a number of machines", *Automation and Remote Control*, Vol. 12, pp. 1972-1981.

[35] Lutz, Christian M.; K. Roscoe Davis; and Minghe Sun (1998), "Determining buffer location and size in production lines using tabu search", *European Journal of Operational Research*, Vol. 106, pp. 301-316.

[36] McCormick, Thomas S.; Pinedo, Michael L.; Shenker, Scott; and Wolf, Barry (1989), "Sequencing in assembly line with blocking to minimize cycle time", *Operations Research*, Vol. 37, No. 6, pp. 925-935.

[37] Michalewicz, Zbigniew (1992), "Genetic Algorithms + Data Structures = Evolution Programs", 3rd edition, Springer-Verlag, New York.

[38] Moccellin, J.V. and M.S. Nagano (1998), "Evaluating the performance of tabu search procedures for flow shop sequencing", *Journal of the Operational Research Society*, Vol. 49, pp. 1296-1302.

[39] Nawaz, M.; Enscore, Emory E. Jr.; and Ham, Inyong (1983), "A heuristic algorithm for the m-machine, n job flow-shop sequencing problem", *Omega International Journal of Management Science*, Vol. 11, No. 1, pp. 91-95.

[40] Negenman, Ebbe G. (2001), "Local search algorithms for the multiprocessor flow shop scheduling problem", *European Journal of Operational Research*, Vol. 128, pp. 147-158.

[41] Norman, Bryan A. (1999), "Scheduling flowshops with finite buffers and sequence dependent setup times", *Computers & Industrial Engineering*, Vol. 36, pp. 163-177.

[42] Nowicki, Eugeniusz (1999), "The permutation flow shop with buffers: A tabu search approach", *European Journal of Operational Research*, Vol. 116, pp. 205-219.

[43] Nowicki, Eugeniusz and Czeslaw Smutnicki (1996), "A fast tabu search algorithm for the permutation flow-shop problem", *European Journal of Operational Research*, Vol. 91, pp. 160-175.

[44] Nowicki, Eugeniusz and Czeslaw Smutnicki (1998), "The flow shop with parallel machines: A tabu search approach", *European Journal of Operational Research*, Vol. 106, pp. 226-253.

[45] Ogbu, F.A. and Smith, D.K. (1990), "The application of the simulated annealing algorithm to the solution of the $n/m/C_{max}$ flowshop problem", *Computers and Operations Research*, Vol. 17, No. 3, pp. 243-253.

[46] Osman, I.H. and Potts C.N. (1989), "Simulated Annealing for permutation flow-shop scheduling", *Omega International Journal of Management Science*, Vol. 17, No. 6, pp. 551-557.

[47] Palmer, D.S. (1965), "Sequencing Jobs through a Multi-Stage Process in the Minimum Total Time - A Quick Method of Obtaining a Near Optimum", *Operational Research Quarterly*, Vol. 16, No. 1, pp. 101-107.

[48] Papadimitriou, Christos H. and Kanellakis, Paris C. (1980), "Flowshop scheduling with limited temporary storage", *Journal of the Association for Computing Machinery*, Vol. 27, No. 3, pp. 533-549.

[49] Pinedo, Michael (1995) "Scheduling: Theory, Algorithms, and Systems", Prentice Hall, Englewood Cliffs, New Jersey 07632.

[50] Rajendran, Chandrasekharan (1994), "A no-wait flowshop scheduling heuristic to minimize makespan", *Journal of the Operational Research Society*, Vol. 45, pp. 472-478.

[51] Reddi, S.S. and Ramamoorthy, C.V. (1972) "On the flow-shop sequencing problem with no-wait in process", *Operational Research Quarterly*, Vol. 23, pp. 323-330.

[52] Reeves, Colin R. (1993), "Improving the efficiency of tabu search for machine sequencing problems", *Journal of the Operational Research Society*, Vol. 44, pp. 375-382.

[53] Reeves, Colin R. (1995), "A genetic algorithm for flowshop sequencing", *Computers and Operations Research*, Vol. 22, No. 1, pp. 5-13.

[54] Reeves, Colin R. (Ed.) (1993), "Modern Heuristic Techniques for Combinatorial Problems", John Wiley & Sons, Inc., New York.

[55] Satake, Tsuyoshi; Katsumi Morikawa; Katsuhiko Takahashi; and Nobuto Nakamura (1999), "Simulated annealing approach for minimizing the makespan of the general job-shop", *International Journal of Production Economics*, Vol. 60-61, pp. 515-522.

[56] Sawik, T. (1993), "A scheduling algorithm for flexible flow lines with limited intermediate buffers", *Applied Stochastic Models and Data Analysis*, Vol. 9, pp. 127-138.

[57] Shaukat, A.B., and Hunsucker, J.L. (1991), "Branch and bound algorithm for the flowshop with multi processors", *European Journal of Operational Research*, Vol. 51, pp. 88-89.

[58] Soewandi, Hanijanto (1998) "Sequencing jobs on the two- and three-stage hybrid flowshop to minimize makespan", PhD. dissertation, North Carolina State University.

[59] Sundararaghavan, P.S.; A.S. Kunnathur; and I. Viswanathan (1997), "Minimizing makespan in parallel flow shops", *Journal of the Operational Research Society*, Vol. 48, pp. 834-842.

[60] Tadei, R.; JND Gupta; F. Della Croce; and M. Cortesi (1998), "Minimizing makespan in the two-machine flow-shop with release times", *Journal of the Operational Research Society*, Vol. 49, pp. 77-85.

[61] Taillard, E. (1990), "Some efficient heuristic methods for the flowshop sequencing problem", *European Journal of Operational Research*, Vol. 47, pp. 65-74.

[62] Turner, Scott and Booth, Dean (1986), "Comparison of heuristic for flowshop sequencing", *Omega International Journal of Management Science*, Vol. 15, No. 1, pp. 75-85.

[63] Vall, Vincente; M. Angeles Perez; and M. Sacramento Quintanilla (1998), "A tabu search approach to machine scheduling", *European Journal of Operational Research*, Vol. 106, pp. 277-300.

[64] Vandevelde, Ann (1994) "Minimizing the makespan in a multiprocessor flow shop", Master's thesis, Eindhoven University of Technology.

[65] Widmer, Marino and Hertz, Alan (1989), "A new heuristic method for the flowshop sequencing problem", *European Journal of Operational Research*, Vol. 41, pp. 186-193.

[66] Wittrock, R.J. (1988), "An adaptable scheduling algorithm for flexible flow lines", *Operations Research*, Vol. 36, pp. 445-453.

[67] Zegordi, Seyed Hessameddin; Kenji Itoh; and Takao Enkawa (1995), "Minimizing makespan for flow shop scheduling by combining simulated annealing with sequencing knowledge", *European Journal of Operational Research*, Vol. 85, pp. 515-531.