DESIGN AND ANALYSIS OF ALGORITHMS FOR SOLVING A CLASS OF ROUTING SHOP SCHEDULING PROBLEMS

LIU SHUBIN

NATIONAL UNIVERSITY OF SINGAPORE
2008

DESIGN AND ANALYSIS OF ALGORITHMS FOR SOLVING A CLASS OF ROUTING SHOP SCHEDULING PROBLEMS

LIU SHUBIN
(M.Eng. NUS)

A THESIS SUBMITTED
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
DEPARTMENT OF INDUSTRIAL & SYSTEMS ENGINEERING
NATIONAL UNIVERSITY OF SINGAPORE
2008

Acknowledgements

First and foremost, I would like to express my sincere gratitude and appreciation to my two supervisors, Associate Professor Ong Hoon Liong and Dr. Ng Kien Ming, for their invaluable advice and patient guidance throughout the whole course of my research. It would be impossible for me to carry out the research work reported in this dissertation without their guidance. In addition, their meticulous attitude towards research and their kind personality will always be remembered. I would also like to take this opportunity to thank all the other faculty members of the department of Industrial & Systems Engineering, from whom I have learned a lot through both coursework and seminars.

Special appreciation also goes to my fellow research students in the department of Industrial & Systems Engineering. Particularly, I am grateful to Han Dongling, Wang Qiang, Zhou Qi, Li Juxin, Sun Hainan, Fu Yinghui, Bae Minju, Chen Liqin, Xing Yufeng, Chang Hongling and Lahlou Kitane Driss, who kindly offered help in one way or another. I would also like to extend my appreciation to those students whose names are not listed here.

Last but not least, special thanks are due to my parents, my wife Zeng Ling, and my son Xin Ji. They gave me continuous encouragement and warm support during the course of my Ph.D. study. This dissertation is dedicated to them.

TABLE OF CONTENTS

Abstract

The role of manufacturing scheduling is to allocate scarce resources to tasks in order to maximize or minimize one or more objectives. Scheduling is a key decision making process and plays an important role in modern manufacturing systems. In modern manufacturing system, the resources may be machines, time, manpower, space, or all of them. In the last four decades, considerable research work have been conducted on classical machine scheduling problems, in which it is often assumed that products can be moved between machines instantaneously, or that machines can travel from one location to another location instantaneously. This assumption may not be valid as it ignores product or machine traveling times, or machine setup times that are inevitable in practice. Therefore, it is necessary to develop machine scheduling algorithms which consider transportation or setup times, in order to reflect real manufacturing scheduling environments better.

By considering transportation times or sequence dependent setup times, the routing shop scheduling problems considered in this research work become an extension of classical shop scheduling problems. As classical shop scheduling problems are special types of routing shop scheduling problems where transportation or setup times are ignored, the algorithms developed for the routing shop scheduling problems can also be applied to the corresponding classical shop scheduling problem where the transportation or setup times are ignored.

In this study, a branch-and-bound algorithm for solving single machine total weighted tardiness problem with unequal release dates was developed. The objective of the problem is to minimize the total weighted tardiness by sequencing the job processing order on a single machine. Three global dominance rules as well as a lower bound computational method were proposed to prune the search tree branches. The

efficiency of the dominance rules and the lower bound computational method were assessed based on comprehensive computational experiments. Our computation results show that the dominance rules and the lower bound are effective in pruning the search tree branches.

In this study, we also developed a general-purpose heuristic, named overlapped neighborhood search (ONS) algorithm, for single machine scheduling problems with or without transportation or setup times. The basic idea of the ONS algorithm is to divide a permutation of the schedule into overlapped blocks; subsequently, the neighborhood of each block is explored independently. The ONS algorithm is also applicable to a wide variety of sequencing problems, such as various single machine scheduling problems, the traveling salesman problem, the linear ordering problem, the quadratic assignment problems, the bandwidth reduction problems and other problems whose solutions can be represented by permutations. The ONS algorithm has been applied to single machine scheduling problems with unequal release dates and the single machine scheduling problem with sequence dependent setup times. The computational experiments carried out in our research work show that the ONS algorithm is efficient in finding near optimal solutions for single machine scheduling problems within reasonable computation times.

The previously mentioned work focuses on single machine scheduling problems. In this research work, heuristics were also developed for two multi-machine scheduling problems, open shop scheduling problems and routing open shop scheduling problems. New neighborhood structures were defined for the two multi-machine scheduling problems. In addition, an exact and fast operation move feasibility checking method was developed for the multi-machine scheduling problems to remove infeasible operation moves quickly. Tabu search algorithms were developed for open

shop and routing open shop scheduling problems based on the new neighborhoods and the new feasibility checking method. To test the performance of the neighborhood structures and the feasibility checking method, comprehensive computational experiments were conducted based on benchmarks and randomly generated problem instances. The computational results show that the tabu search algorithms embedded with the new neighborhoods are able to find optimal or near optimal solutions for most of the problem instances tested, within reasonable computation times.

List of Tables

List of Figures

List of Symbols

JSSP	Job shop scheduling problem
OSSP	Open shop scheduling problem
FSSP	Flow shop scheduling problem
ROSSP	Routing open shop scheduling problem
RSSP	Routing shop scheduling problem
TSP	Traveling salesman problem
VRP	Vehicle routing problem
SMSP	Single machine scheduling problem
r_j	The release date of job j
w_j	The weight of job j , which is a priority factor to denote the importance of job j . It also denotes the tardiness penalty factor for tardiness related problem.
C_j	Completion time of job j
T_j	Tardiness of job j
C_{max}	The makespan of a machine scheduling problem
O_{ij}	Operation of job J_j processed on machine i
$C_{min}(S)$	The minimum completion time for the jobs in set S
$C_{max}(S)$	An upper bound of maximum completion time for the optimal schedules
$IPM(i)$	The immediate predecessor of operation i on machine M_i in a schedule
$ISM(i)$	The immediate successor of operation i on machine M_i in a schedule
$IPJ(i)$	The immediate predecessor of operation i belonging to job J_i in a schedule
$ISJ(i)$	The immediate successor of operation i belonging to job J_i in a schedule
h_i	The head of operation i, which is the longest path from source node to node i
t_i	The tail of operation i, which is the longest path from sink node to node i

Chapter 1 Introduction

This dissertation focuses on the design and analysis of algorithms for solving a class of routing shop scheduling problems. In the last four decades, considerable research work has been carried out on manufacturing scheduling problems, where different jobs are sequenced in order to optimize one or more criteria. However, most of the previous research work focused on classical shop scheduling problems without considering transportation times of the semi-finished product, traveling times of machines (in the case where the machines need to travel from job to job) or sequence dependent machine setup times. By considering transportation times or sequence dependent setup times, the scheduling problems considered in this research work become an extension to the classical shop scheduling problems.

1.1 Background

The role of scheduling is to allocate scarce resources to tasks over time to maximize or minimize one or more objectives. As pointed out by Pinedo (2002), the resources and tasks can take many forms depending on the type of organization, e.g. personnel, space and time in a restaurant, processing power of a server, machines and raw material in a manufacturing company and so on. The objective of the scheduling problem is to assign machines and resources to jobs in order to complete all jobs under the pre-specified constraints to optimize one or more criteria.

In the last four decades, considerable research work was carried out on the classical machine scheduling problem. The classical machine scheduling problem normally assumes that there are an infinite number of transport vehicles and that the semi-finished products can be delivered instantaneously from one location to another.

Realistically, in typical manufacturing environments, this assumption is not valid as the semi-finished products have to take some time to be delivered from one location to another. In some cases, such as engine casings of ships, the parts are too big or too heavy to be moved between machines and hence the machines have to travel between jobs (Averbakh and Berman 1996). Another example is the scheduling of robots that perform daily maintenance of operations on immovable machines located in different locations (Averbakh and Berman 1999). The machine scheduling problem that takes transportation or setup times into consideration is referred to as the routing shop scheduling problem (RSSP). The RSSPs include both single machine scheduling problems and multi-machine scheduling problems. It is noted that transportation times and sequence dependent setup times are considered equivalent to each other for the RSSPs as the two types of problems can be tackled in the same way.

1.2 Overview of General Solution Methodology

The general algorithms that are applicable to many different types of machine scheduling problems can be classified into six categories. They are dispatching rules, mathematical programming, branch-and-bound techniques, neighborhood based local search algorithms, artificial intelligence, and constraint programming techniques, respectively.

Dispatching rules, which are also called priority rules, are probably the most frequently applied heuristics for solving machine scheduling problems in practice because of their ease of implementation and low requirements on computational power.

Most of the machine scheduling problems are combinatorial optimization problems. One of the most popular solution techniques for combinatorial optimization problems is branch-and-bound. The principle of the branch-and-bound technique, as

described by Agin (1966), is the enumeration of all feasible solutions of the problem. The basic idea of branching is to conceptualize the problem as a decision tree with each branch defining a subset of all feasible solutions of the original problem. The decision tree grows until leaf nodes, which cannot branch further, are reached. In general, the union of the subsets of solutions at the same depth level is equal to the set of the original problem's feasible solutions and there is no intersection with each other. To speed up the enumeration procedure, the objective value of the best solution from a subset is estimated as the lower bound for a minimization problem. Whenever the lower bound is equal to the best known upper bound, the branch is pruned from further consideration. For integer programming formulation of the machine scheduling problem, the Lagrangian relaxation technique described by Shapiro (1979) can be used to solve the relaxed problem by omitting certain specific integer-value constraints to obtain a lower bound.

A local search algorithm starts from an initial candidate solution and then iteratively moves to a neighboring solution based on a pre-defined neighborhood space. Typically, every candidate solution has more than one neighboring solution and the choice of which one to move to is based only on information found in the neighborhood of the current solution.

Constraint programming (CP) is a relatively new technique for solving combinatorial optimization problems in the computer science community. Constraint programming is based on finite domains and is particularly suited to combinatorial optimization problems as it is an assignment of values to variables such that a set of constraints on variable pairs are satisfied as claimed by Minton et al. (1992).

Artificial intelligence (AI) techniques have been applied to job shop scheduling problems (JSSP) since the early 1980s. AI techniques include the use of expert systems,

knowledge-based systems and several other techniques. AI techniques have four main advantages compared with other methods, as stated by Jones and Rabelo (1998). First, AI techniques use both quantitative and qualitative knowledge in the decision making process. Second, they generate solutions using complex heuristics rather than simple dispatching rules. The third advantage is that AI techniques select the heuristic depending on the entire scheduling decision-making related information. The final advantage is that they can capture complex relationships in elegant new data structures and contain some unique techniques to manipulate the information in these data structures. However, AI techniques have two serious disadvantages. Firstly, an AI system is difficult to be built, implemented and maintained. Secondly, it is difficult to evaluate the closeness of the solutions generated using AI techniques to the optimal solutions.

1.3 Motivation and Purpose of this Study

It is commonly assumed that transportation times can be ignored or that setup times are independent of the job processing sequence. However, significant setup times may elapse in situations where the machine is setup to process different types of jobs. Many practical industrial situations require consideration of transportation or setup times. These situations can be found in various environments, such as in production, services industry, and information processing. As stated by Lee and Chen (2001), the coordination of manufacturing and distribution systems must be made carefully in order to achieve ideal overall system performance. It is also obvious that to reflect a realistic manufacturing system, machine scheduling problems that consider transportation or setup times are superior to classical machine scheduling problems that do not take these times into account.

The purpose of this study is to design and analyze algorithms for solving a class of RSSPs. The RSSPs that consider transportation or setup times are able to reflect realistic machine scheduling systems better than classical machine scheduling problems. Therefore, it is possible to design algorithms that are able to improve the overall system performance by considering both the job processing times, and the transportation or setup times.

In this research, we first consider the single machine total weighted tardiness problem with unequal release dates. Then, a new general-purpose heuristic, named overlapped neighborhood search (ONS) algorithm, is presented to solve the general single machine scheduling problems. Finally, we propose new neighborhood structures for multi-machine scheduling problems with and without transportation times.

1.4 Organization of this Dissertation

This dissertation is organized as follows. In Chapter 2, a literature review of the algorithms developed for the machine scheduling problems is presented. A branch-and-bound algorithm is proposed for the single machine total weighted tardiness problem with unequal release dates in Chapter 3. In Chapter 4, we present a brand new heuristic, called overlapped neighborhood search algorithm, for the general sequencing problems whose solutions can be represented by permutation. New neighborhood structures are defined for both the open shop scheduling problem (OSSP) and the routing open shop scheduling problem (ROSSP) in Chapter 5. Tabu search algorithms that are based on existing and new neighborhoods are presented for both the OSSP and ROSSP. In Chapter 6, the summary, conclusions and suggestions for future research are provided.

Chapter 2 Literature Review

In this chapter, heuristic and exact algorithms developed for both classical machine scheduling problems and RSSPs are reviewed. We first give a review of the general algorithms developed for the machine scheduling problem. Then, a detailed review is presented for classical machine scheduling problems and RSSPs based on the optimization criteria.

2.1 Classification of Machine Scheduling Problems

The scheduling problems are generally denoted by the three-field scheduling notation $\alpha \mid \beta \mid \gamma$ proposed by Graham et al. (1979) and extended by Błażewicz et al. (2001). The first field denotes the machine environment and contains a single entry. The second field provides details of the processing characteristics and the constraints, and may contain no entries or multiple entries. The third field describes the objective to be optimized and usually contains one entry. In the scheduling problems considered in this research work, the number of jobs and machines are assumed to be finite, and are denoted by n and m respectively. Usually, we use j to denote a job and i to denote a machine. If a job requires a number of operations to be completed, then the pair (i, j) refers to the operation of job j to be processed on machine i.

Some machine environments (specified in the field α) that have been studied in literature are summarized below.

Single machine (1 **)**	Only one machine in this problem, it is a special case of all other more complicated problems.
Identical machines in parallel (P_m **)**	There are n single-operation jobs and m identical machines. Each job may be processed on one or more machines but can only be processed on one machine at a

	time.
Parallel machines with different speeds, also called uniform machines (Q_m)	There are m parallel machines with different speeds. The time spent to process job j is $p_{ij} = p_j / v_i$, where p_j is the standard processing unit of job j and v_i is the speed of machine i.
Unrelated machines in parallel (R_m)	There are m machines in parallel with processing speed of v_{ij} if job j is processed on machine i. The time spent to process job j is $p_{ij} = p_j / v_{ij}$
Flow shop (F_m)	There are m parallel machines. A job consists of a collection of operations and all jobs will follow the same route.
Open shop (O_m)	There are m machines. Each job has to be processed on each of the m machines. There are no restrictions on the routing of each job through the machine environment.
Job shop (J_m)	In a job shop environment with m machines, a job consists of a collection of operations that have a predetermined route to follow.

The processing constraints specified in the field β may contain more than one entry.

Some machine environments are given below.

Release dates (r_j)	If r_j does not appear in the β field, the processing of job j may start at any time; otherwise the job cannot be processed before its release date r_j.
Preemptions (*prmp*)	Preemption denotes that a job can be stopped from processing before its completion and its processing can be resumed later.
Precedence constraints (*prec*)	For single machine and parallel machine environments, one or more jobs have to be completed before another job is allowed to be processed.
Sequence dependent setup times (s_{jk})	The s_{jk} indicates the sequence dependent setup time between jobs j and k. If s_{jk} does not appear in field β, all setup times are assumed to be 0 or sequence independent.
Permutation (*prmu*)	This is a constraint that may appear in a flow shop environment. It restricts the queues in front of each machine to operate according to the First in First out (FIFO) rule. The *prmu* constraint implies that the sequence in which the jobs are processed in the first machine is maintained throughout the system.
Machine eligibility restrictions (M_j)	The M_j symbol in the P_m environment in the field β implies that only machines in the set M_j can process the job j. If M_j does not appear in the P_m environment, it means the job j can be processed on any machine.

Recirculation (*recrc*)	Recirculation will occur in a flexible job shop or job shop when a job is required to visit a machine more than once.
No wait (*no-wait*)	Buffers at the machines have zero capacity and a job, upon finishing its processing on one machine, must immediately start on the next machine.

For the scheduling problem, the objectives to be optimized are always functions of the completion times of the jobs. The completion time of the job j on the machine i is denoted by C_{ij}. The objective may also be a function of due date d_j. The lateness of a job is defined as

$$L_j = C_j - d_j .$$

It is obvious that L_j is negative if the corresponding job is completed late and positive if completed early. The tardiness of job j is defined as

$$T_j = \begin{cases} C_j - d_j & \text{if } C_j > d_j \\ 0 & \text{otherwise} \end{cases} .$$

Another due date related penalty function is whether a job is late or not. It is defined as

$$U_j = \begin{cases} 1 & \text{if } C_j > d_j \\ 0 & \text{otherwise} \end{cases} .$$

The objectives of scheduling problems are divided into two classes, namely, regular measures of performance and non-regular measures of performance. For the regular measures of performance, the objective value is nondecreasing with job completion times. That is to say, if any job is made to finish later, the measure, for e.g., flow time, makespan, lateness, tardiness, etc, will stay the same or increase. The non-regular measures of performance evaluate the objectives other than the regular measures of performance. An example is the sum of earliness and tardiness penalties, where the larger the deviation, the larger the penalty. For the regular measures of

performance, there always exists an active schedule that is optimal (Baker 1974). Some objectives of the scheduling problem to be minimized are summarized below.

Makespan ($C_{\max}$)	The makespan is the maximum completion time of all jobs in the system. It is defined as $C_{\max} = \max\{C_1, \ldots, C_n\}$.
Maximum lateness ($L_{\max}$)	The maximum lateness $L_{\max}$ is a measure of the worst violation of due dates among all the jobs, which is defined as $L_{\max} = \max\{L_1, \ldots, L_n\}$.
Maximum tardiness ($T_{\max}$)	The maximum tardiness is equivalent to $L_{\max}$ when $L_{\max} \geq 0$, $T_{\max} = 0$ when $L_{\max} < 0$. The maximum tardiness is defined as $T_{\max} = \max\{T_1, \ldots, T_n\}$.
Total weighted completion time ($\sum w_j C_j$)	The sum of the completion times is known as the flow time. $\sum w_j C_j$ is also called the weighted flow time. If w_j denotes the inventory holding cost and C_j denotes the holding time, the total weighted completion time indicates the total holding cost.
Total weighted tardiness ($\sum w_j T_j$)	The total weighted tardiness objective function is to minimize the total weighted tardiness of the tardy jobs.
Total earliness penalty ($\sum E_j$)	The objective is called non-regular objective as its objective value is nonincreasing with respect to C_j.

Before the end of this subsection, we give a classification of the type of schedules defined by Baker (1974).

A feasible schedule is called a semi-active schedule if no operation can be started earlier without altering the order of jobs on any machine. An active schedule is a schedule in which no operations can be relocated to a position to complete earlier without delaying other operations. A schedule is defined as a non-delay schedule if no machine is kept idle at a time when at least one operation is available for processing. The relationship of the three types of schedules is illustrated in Figure 1.1.

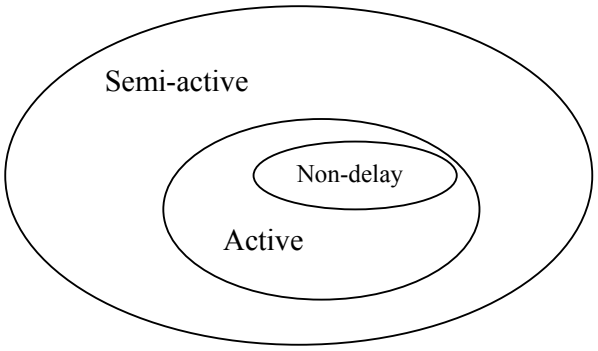


Figure 1.1 The relationship of the three types of schedules

2.2 Algorithms for Classical Machine Scheduling Problem

Machine scheduling is concerned with scheduling computer or manufacturing processes because the same model and algorithm can be applied to the two different areas. In the last four decades, a lot of research work was done on deterministic and stochastic machine scheduling problems and an astounding number of machine scheduling problems have been defined. For different kinds of problems, many exact and heuristics can be found in the literature. As it is impossible to give a detailed review of all machine scheduling problems in this dissertation, this review will focus on the deterministic single machine problem, the open shop problem, the flow shop problem and the job shop problem. As there are different objective criteria for each type of scheduling problem, we will only concentrate on the models and algorithms that aim to minimize the makespan and total weighted tardiness. For a complete review of machine scheduling problems, models and algorithms, the reader is referred to Baker (1974), Błażewicz et al. (2001), and Pinedo (2002).

2.2.1 Single Machine Scheduling Problem

This subsection reviews the single machine scheduling problem, which is the simplest scheduling problem with only one machine available. The models and algorithms developed for the single machine scheduling problem not only provide insights into this problem but also provide a basis for more complicated scheduling problems, such as the JSSP, the FSSP and the OSSP. This subsection is organized as follows. The models and algorithms for minimizing makespan are reviewed for different types of single machine scheduling problems. Then it is followed by the models and algorithms that were developed for minimizing the total weighted tardiness.

Minimizing the Makespan

The makespan of a single machine scheduling problem is the maximum completion time of all jobs. The problem of minimizing makespan is one of the simplest machine scheduling problems and polynomial algorithms are available for some of these problems.

Problem $1 \mid r_j \mid C_{\max}$

For problem $1 \mid r_j \mid C_{\max}$, the optimal solution can be obtained by ordering the jobs in nondecreasing order of release dates. When the release dates for all the jobs are zero, the above problem is reduced to $1 \mid\mid C_{\max}$ with the makespan $C_{\max} = \sum_{j=1}^{n} p_j$.

Problem $1 \mid r_j, \tilde{d}_j \mid C_{\max}$

This problem is to minimize the makespan with a specified release date r_j and deadline $\tilde{d}_j$ for each job. This problem is NP-hard in the strong sense as proven by Lenstra et al. (1977). Bratley et al. (1973) developed a branch-and-bound algorithm

based on the implicit enumeration of a search tree for this problem. From the root node of the search tree, n new branches are generated at the first level of the descendant nodes. Assume J_j is at the i^{th} node in level k, it represents the job J_j sequenced at the k position in the schedule. It is evident that all the $n!$ possible schedules have to be enumerated following the search tree. To reduce the number of nodes to be searched, the following node elimination criteria are used.

(1) Exceeding deadline. If a job is scheduled at a level with the completion time exceeding its deadline, we know that this schedule is infeasible and this node is fathomed.

(2) Problem decomposition. Consider a job J_j, which is scheduled at level k. If the completion time of J_j is greater than or equal to the earliest release date of the unscheduled jobs, then there is no need to enter another branch at level k. The reason for this node elimination feature is that the best schedule for the remaining jobs may not be started prior to the earliest release date, and hence cannot complete earlier than the completion time of J_j. From another point of view, since the active schedules contain at least one optimal schedule, assigning other unscheduled jobs before job J_j will generate a non-active schedule that is not needed.

Problem $1 \mid s_{jk} \mid C_{\max}$

In many manufacturing environments, the setup times depend on the type of job that is just completed as well as on the job to be processed. Sequence dependent setup times are commonly found where a single machine is the resource used to produce different kinds of products. This type of problem is often interpreted as a traveling salesman problem (TSP) as claimed by Baker (1974). The setup time s_{jk}

corresponds to the distance between two nodes *j* and *k*. It is noted that in an asymmetric TSP, s_{jk} may not be equal to s_{kj}.

Minimizing the Total Weighted Tardiness

In this subsection, we review those problems whose objective is to minimize the total weighted tardiness, which is equivalent to the mean weighted tardiness. The objective is to measure the time-dependent penalties on late jobs but without any benefits derived from completing the jobs early. For a recent complete review of the single machine weighted tardiness problem, the reader can refer to Abdul-Razaq et al. (1990) and Sen et al. (2003).

Problem $1||\sum T_j$

The problem $1||\sum T_j$ has attracted many researchers and received an enormous attention in the literature. Its complexity was open until Du and Leung (1990) proved that $1||\sum T_j$ is NP-hard in the ordinary sense. Many algorithms, including priority rule based heuristics, local search and branch-and-bound, have been proposed to deal with problem $1||\sum T_j$. The simplest rules are shortest processing time (SPT) and earliest due date (EDD) rules. Montagne (1969) proposed a rule in which the jobs are sequenced in nonincreasing order of $p_j/(\sum_{i=1}^{n} p_i - d_j)$. Baker and Bertrand (1982) developed a dynamic implementation of the EDD rule based on modified due dates (MDD). The MDD rule is to schedule the jobs dynamically according to the earliest MDD, where $MDD = \max\{C + p_i, d_i\}$, with C being the completion time of the last scheduled job. Rachamadugu and Morton (1981) proposed an apparent urgency (AU) rule, in which the priority is defined as $AU_j = (1/p_j)\exp\left(-\max\{0, d_j - t - p_j\}/k\bar{p}\right)$,

where k is called the lookahead parameter which is set according to the tightness of the due date, $\bar{p}$ is the average processing time, and t is the current time. It is noted that the priority rules developed for problem $1||\sum T_j$ can also be extended to solve other single machine tardiness problems.

Wilkerson and Irwin (1971) proposed a heuristic based on the idea of adjacent pairwise interchange. The conditions under which the jobs with earlier due dates should be scheduled earlier are identified. Adjacent pairwise interchange will be carried out if the conditions are violated. Fry et al. (1989) developed an adjacent pairwise interchange based local search method. Holsenback and Russell (1992) presented a net benefit of relocation (NBR) heuristic method, where a job will be relocated if the net change in tardiness due to the relocation of the job is negative. Potts and Van Wassenhove (1991) proposed a simulated annealing based meta-heuristic method.

Problem $1|r_j|\sum T_j$

Chu and Portmann (1992) showed that problem $1|r_j|\sum T_j$ can be simplified by using a modified due date and a branch-and-bound method was developed based on the modified due date. Baptiste et al. (2004) developed a tighter lower bound than that defined in Chu and Portmann (1992) for problem $1|r_j|\sum T_j$.

Problem $1||\sum w_j T_j$

Problem $1||\sum w_j T_j$ is NP-hard in a strong sense as shown by Lenstra et al. (1977). This problem is extended from problem $1||\sum T_j$ by considering different weights. Therefore the priority rules for problem $1||\sum T_j$ can all be extended to solve problem $1||\sum w_j T_j$ by considering the weights. It is noted that the algorithms

developed for weighted tardiness problems may not always be effective methods for equal weighted problems because the latter has some special structures. Branch-and-bound algorithms have been developed by Elmaghraby (1968), Rinnooy et al. (1975), and Potts and Van Wassenhove (1985) for problem $1||\sum w_j T_j$.

Problem $1|r_j|\sum w_j T_j$

Akturk and Ozdemir (2000) developed the first branch-and-bound approach for problem $1|r_j|\sum w_j T_j$. Another branch-and-bound algorithm was proposed by Jouglet et al. (2004). In addition to the exact approaches, a local dominance rule was presented by Akturk and Ozdemir (2001) for problem $1|r_i|\sum w_i T_i$ and a pairwise improvement heuristic was proposed by Chou (2005). Jouglet et al. (2008) introduced a tabu search algorithm based on dominance rules.

Problem $1|s_{jk}|\sum w_j T_j$

Raman et al. (1989) gave a method derived from the apparent tardiness rule (ATC) to take setup times into account. Ragatz (1993) presented a branch-and-bound algorithm to minimize the total tardiness with sequence dependent setup times. The lower bounds used by them are weak and the performance of the procedure is highly dependent on the problem's characteristics. Rubin and Ragatz (1995) developed a genetic algorithm where a local search method based job exchange or single random exchange was embedded to improve the solutions produced by the genetic operators further. Tan and Narasimhan (1997) applied simulated annealing to minimize the total tardiness on a single machine with sequence dependent setup times. A comparison work of the four algorithms, namely branch-and-bound, genetic algorithm, simulated annealing, and random start pairwise interchange was conducted by Tan et al. (2000). França et al. (2001) proposed a mimetic algorithm (MA), in which a new solution was

generated from the genetic operators of selection, crossover and mutation. A local search method was then applied to the new solution to improve it further. Gagné et al. (2002) presented an ant colony optimization (ACO) algorithm for the single machine scheduling problem with sequence dependent setup times.

Gupta and Smith (2006) presented two algorithms, a problem space-based local search heuristic, and a GRASP algorithm with path relinking for the single machine scheduling problem with sequence dependent setup times. The authors claimed that the space-based local search method performed equally well as the ACO algorithm and that the GRASP gave better solutions than the ACO in general. Armentano and de Araujo (2006) proposed several variants of the GRASP based algorithm by incorporating memory-based mechanisms.

2.2.2 Flow Shop Scheduling Problem

In many manufacturing environments, it is required that a number of operations be processed on every job. If the machines are assumed to be set up in series and the operations have to be done on all jobs in the same order, then the manufacturing system is referred to as a flow shop. A typical FSSP consists of *m* machines that perform operations on all the *n* jobs. There are several constraints for the FSSP.

(1) There are no precedence relations among the operations of different jobs;

(2) Each machine can perform only one operation at a time and cannot be interrupted;

(3) Each job can only be processed on one machine at a time.

Most of the research conducted on FSSP is limited to a special case of the flow shop, called the permutation flow shop, where the jobs are processed on each machine in the same order. For the permutation FSSP, once the job sequence on the first

machine is fixed, it becomes fixed for the other machines as well. The typical objectives of FSSP include minimizing the makespan, minimizing the average flow time, and minimizing the mean tardiness or the number of tardy jobs. A brief review is given in the following subsection for the permutation flow shop problem. A recent complete review for FSSPs can be found in Framinan (2004), and Kis and Pesch (2005), where exact algorithms are provided.

Minimizing the Makespan

Minimizing makespan for the FSSP is shown to be NP-hard by Garey et al. (1979), except for some special cases. The two-machine flow shop case, $F2 \,||\, C_{\max}$ is easy and a polynomial algorithm was developed by Johnson (1954).

As $F_m \,||\, C_{\max}$ is NP-hard for $m \geq 3$, many researchers focused on the development of priority rules and local search based heuristics. The heuristic method developed by Campbell et al. (1970) obtains a complete schedule by solving *m* - 1 of a two-machine approximation, based on Johnson's algorithm (Johnson 1954). Gupta (1971) generalized Johnson's two-machine flow shop algorithm to solve the FSSP with more than three machines. The index of job J_j is defined as,

$$s_j = \lambda / \min_{1 \leq i \leq m-1} \{p_{ij} + p_{i+1,j}\} \; for \quad j = 1, \ldots, n \text{, where } \lambda = \begin{cases} 1 & if \;\; p_{ij} \leq p_{1j} \\ -1 & otherwise \end{cases}.$$

Ho and Chang (1991) presented an improved heuristic method by minimizing the gaps between successive operations in a schedule. Tabu search algorithms for the flow shop problem were developed by Widmer and Hertz (1989), and Taillard (1990) independently. The neighbors are defined similar to those in the TSP as given below.

(1) Exchange two adjacent jobs;

(2) Exchange the jobs placed at two different positions;

(3) Move a job to another position.

Some other heuristic methods for the FSSP are simulated annealing by Osman and Potts (1989), and genetic algorithm developed by Reeves (1995). Huang and Wang (2006) proposed a local search method with escape-from-trap procedures for the FSSP.

Minimizing the Total Tardiness

The problems of minimizing average flow time and due date related objectives tend to be even harder. Compared to the algorithm for minimizing the makespan, there are fewer algorithms for minimizing the total tardiness of a FSSP. Onwubolu and Michael (1999) developed a genetic algorithm considering three objectives, namely minimizing the total tardiness, minimizing the makespan, and minimizing the number of tardy jobs. Pan et al. (2002) proposed a branch-and-bound algorithm to minimize the total tardiness for a two-machine FSSP. Hasija and Rajendran (2004) presented a simulated annealing method in which two perturbation schemes and a new improved scheme were incorporated to to minimize the total tardiness. Kyparisis and Koulamas (2006) presented an algorithm considering all the regular objective functions with tight worst-case performance bounds by utilizing the optimal permutations for the corresponding single machine problems.

2.2.3 Job Shop Scheduling Problem

A JSSP consists of a set of different machines on which the operations of each job are processed. Unlike the FSSP and OSSP, the JSSP has a specific operation processing order for each job. The three constraints of FSSP detailed in subsection 2.2.2 also apply to the JSSP. The JSSP is a well-known NP-hard problem; see Lenstra et al. (1977). The JSSP has received enormous attention and considerable research

papers on it have been published. In this subsection, we will give a review of the JSSP algorithms for minimizing makespan and total weighted tardiness. The survey papers on JSSP techniques were given by Panwalker and Iskander (1977), Blackstone et al. (1982), Haupt (1989), Vaessens et al. (1996), Błażewicz et al. (1996), and Jones and Rabelo (1998).

Minimizing the Makespan

The techniques for minimizing the makespan for a JSSP are classified into four categories, namely priority rules, heuristics, exact methods, and artificial intelligence based methods. It must be noted that the techniques for solving JSSPs are not limited to the four categories as research work is still very active.

Priority rules

Priority rules are the most widely applied technique in practice because of their ease of implementation and low requirements of computational power. Most of the priority rules are based on the active schedule generation algorithm proposed by Giffler and Thompson (1960). Their algorithm always tries to assign the currently available operations to a machine and conflicts are resolved randomly. The survey paper on scheduling priority rules can be found in Panwalker and Iskander (1977), Blackstone et al. (1982), and Haupt (1989). Some commonly used rules are summarized in Table 2.1.

Heuristic Search Methods

The shifting bottleneck procedure (SBP), proposed by Adams et al. (1988), is one of the most powerful procedures among all heuristic methods developed for the JSSP.

The idea of the SBP is to solve a $1||L_{\max}$ problem for each machine to optimality under the assumption that the optimal schedule sequences of problems $1||L_{\max}$ coincide with an optimal JSSP schedule. Based on the idea of SBP, Dauzere-Peres and Lasserre (1993) proposed a modified shifting bottleneck procedure for the JSSP.

Table 2.1 Priority rules

Rule	Description
1. SPT (shortest processing time)	Select the job with the shortest processing time
2. FCFS (first come first served)	Schedule the first operation waiting in the queue first
3. Random	The next operation is selected randomly
4. SRPT (shortest remaining processing time)	Select the operation with the shortest remaining job processing time
5. LPT (longest processing time)	Select the job with the longest processing time
6. SOT(shortest operation time)	Select the operation with the shortest processing time for the machine being considered
7. LOT (longest operation time)	Select the operation with the longest processing time for the machine being considered

Different neighborhoods were also devised and simulated annealing methods were developed by Matsuo et al. (1988) and van Laarhoven et al. (1992). Based on the same neighborhood definition, tabu search based heuristic procedures were proposed by Dell'Amico and Trubian (1993), Taillard (1994), Barnes and Chambers (1994), Sun et al. (1995), Nowicki and Smutnicki (1996), and Balas and Vazacopoulos (1998). Another meta-heuristic method, genetic algorithm, is also applied to the JSSP. Nakano and Yamada (1991) proposed an encoding method for the JSSP by using a 0-1 matrix to present a solution. Dorndorf and Pesch (1995) presented an encoding method by interpreting an individual solution as a sequence of decision rules. Bierwirth (1995)

introduced a representation of an individual as a string of length equal to the number of operations in the problem. Gonçalves et al. (2005) presented a hybrid genetic algorithm, in which the chromosome representation of the problem is based on random keys.

Unlike traditional heuristic methods, some multi-agent based algorithms used in the field of artificial intelligence have been proposed in order to solve the JSSP. Ghedira and Ennigrou (2000) proposed an algorithm to solve the JSSP based on the cooperation of different agents. A negotiation based scheme was developed by MacChiaroli and Riemma (2002) to make scheduling decisions based on the multi-agent system. In the paper by Aydin and Fogarty (2004), autonomous agents cooperated by sharing solutions via a common-memory. Caridi and Cavalieri (2004) gave a review of multi-agent systems for production planning.

Exact Methods

Considerable amount of work have been done to develop efficient exact algorithms for the JSSP. JSSPs have been formulated using integer programming by Manne (1960) and Balas (1969, 1985), and using mixed integer programming by Adams et al. (1988). A survey of the mathematical programming models was given by Błażewicz et al. (1991). A polynomial lower bound computation method was first proposed by Balas (1985). The branch-and-bound algorithm developed by Carlier and Pinson (1989) is based on a polynomial lower bound obtained for the single machine problems with precedence constraints, task arrival times, and allowed preemptions. Some other efficient branch-and-bound methods have been developed by Applegate and Cook (1991), Brucker et al. (1994), and Martin and Shmoys (1996).

In recent years, constraint propagation techniques were shown to be highly effective for solving the JSSP, such as the algorithm developed by Caseau and Laburthe (1995), Nuijten and Le Pape (1998), and Sourd and Nuijten (2000). A comprehensive summary of constraint-based methods for scheduling can be found in Baptiste et al. (2001).

Minimizing the Total weighted Tardiness

In contrast to the numerous publications on $J_m \| \sum C_{\max}$, only several papers on the $J_m \| \sum w_j T_j$ problem have been published. Pinedo and Singer (1999) presented a heuristic for problem $J_m \| \sum w_j T_j$ based on the famous shifting bottleneck heuristic method of Adams et al. (1988). The critical machine is identified based on the solution of a modified ATC scheduling rule for the JSSP. A computational study of branching techniques for problem $J_m \| \sum w_j T_j$ was conducted by Singer and Pinedo (1998). Theoretical problem difficulty analysis based on statistical methods was carried out to analyze the properties of the problem instances. Asano and Ohta (2002) considered problem $J_m \| \sum w_j T_j$ and proposed a heuristic based on a tree search procedure, in which a JSSP to minimize the maximum tardiness, subject to fixed sub-schedules, was solved at each node of the search tree. Mason et al. (2005) presented a mixed integer programming (MIP) model to minimize the total weighted tardiness for a complex JSSP.

2.2.4 Open Shop Scheduling Problem

OSSP is similar to FSSP except that there are no operation precedence relationships for any job. The OSSP can be formally described as follows: there are a

set of jobs and the operations of each job have to be processed on different machines, within a fixed duration, without any restrictions on the operation processing order. In addition, each machine can only process at most one operation at a time and an operation cannot be interrupted once started. This problem is also called a non-preemptive OSSP. The two common objectives of an OSSP are minimizing the maximum finish time of operations (or makespan), and minimizing the total tardiness. A survey on the recent research achievements on the OSSP can be found in the paper by Dorndorf et al. (2001).

Minimizing the Makespan

The problem of minimizing the makespan is denoted as $O \| C_{\max}$ according to the classification by Graham et al. (1979). A schedule for an OSSP is an assignment of the operations to the machines with the operation processing order on each machine and the processing order of the operations belonging to the same job. The OSSP is similar to the JSSP with the exception that there is no processing order restriction placed on the operations that belong to the same job. Therefore, the OSSP has a larger solution space compared to the JSSP. Similar to the JSSP, the OSSP is also a NP-hard problem (Garey and Johnson 1979). However, it has also been shown that some specially structured OSSPs with $m \geq 3$ are polynomially solvable (Fiala 1983).

Compared with the JSSP, the OSSP received less attention in the mathematics and operations research communities. When there are only two machines, that is, $m = 2$, a polynomial computational complexity algorithm was developed by Gonzalez and Sahni (1976). A shifting bottleneck procedure was presented by Ramudhin and Marier (1996) for the general OSSP based on the shifting bottleneck procedure (SBP) for the JSSP proposed by Adams et al. (1988). Brucker et al. (1997) proposed a

branch-and-bound algorithm that is based on the disjunctive graph formulation of the OSSP.

Two dispatching rule based heuristics were developed by Guéret and Prins (1998) and the performance of the two heuristics was evaluated based on both randomly generated problem instances as well as benchmark problem instances. The first of their heuristic is a list scheduling algorithm with two different priorities while the second heuristic is based on the construction of matchings in a bipartite graph. Liaw (1998) developed an iterative improvement approach based on Benders' decomposition, in which the sequencing and scheduling of operations were attacked individually. A neighborhood structure for the OSSP was proposed by Liaw (1999) and a tabu search algorithm was proposed based on the neighborhood structure proposed by the author. The performance of the tabu search algorithm was evaluated based on both randomly generated problem instances as well as benchmark problem instances. It was claimed by the author that the tabu search algorithm performed extremely well on all the test problems. The same author also developed a hybrid algorithm by incorporating the tabu search algorithm with a genetic algorithm (Liaw 2000). Pinedo (2002) developed a dispatching rule called Longest Alternative Processing Time first (LAPT) rule for problem $O_2 \,||\, C_{\max}$, which was able to produce an optimal solution within polynomial computation time. Another exact branch-and-bound algorithm was developed by Dorndorf et al. (2001). This branch-and-bound algorithm focuses on constraint propagation based methods to reduce the search space. Puente (2004) described an algorithm that combined heuristic rules and genetic algorithms. Senthilkumar and Shahabudeen (2006) presented a genetic algorithm based heuristic for the OSSP. The performance of their algorithm was tested on small sized, randomly generated problems. An exact algorithm was proposed by Tamura et al. (2006) by encoding

Constraint Satisfaction Problems (CSPs) and Constraint Optimization Problems (COPs) with integer linear constraints into Boolean Satisfiability Testing Problems (SATs). The performance of their algorithm was evaluated based on benchmark problem instances. This algorithm found and proved 192 optimal solutions from among the 194 problem instances.

Minimizing the Tardiness

A tabu search approach was proposed by Liaw (2003) when considering a two-machine preemptive OSSP $O_2 \mid prmp \mid \sum T_j$, in which a linear programming model is used to generate the optimal job completion times and a tabu search approach was then applied to generate the schedule. Liaw (2005) developed a branch-and-bound method to solve the $O_m \mid prmp \mid \sum T_j$ optimally.

2.3 Algorithms for Routing Shop Scheduling Problem

In classical shop scheduling models, it is normally assumed that the jobs can be moved between the machines instantaneously. However, this assumption is seldom valid, as it ignores the product or machine transportation times, which are, in practice, not negligible. The scheduling problem that takes into account these transportation times is called a routing shop scheduling problem. Two models are used by Lee and Strusevich (2005) to incorporate transportation times into the flow shop and open shop scheduling problems. In the first model, the machines are located at fixed positions and jobs move between the machines. It is therefore reasonable to assume that there is a time lag between the completion time of a job on one machine and its subsequent start time on another machine. This lag is called transportation time. In the second model, it is assumed that jobs which are located at the nodes of some transportation network and

the machines have to travel between the jobs. It is also assumed that all the machines are located at the same node (depot) initially and have to return to the depot after all jobs are completed. The RSSP is NP-hard as even its relaxed problem, the TSP, is already NP-hard. Most publications in the literature concentrate on the special RSSPs, such as two-machine flow shop and two-machine open shop problems, because of their complexity. A review of some of the RSSPs was made by Lee and Chen (2001).

2.3.1 Single Machine Scheduling Problem with Transportation Times

In a single machine scheduling problem with sequence dependent setup times, n jobs have to be sequenced on a machine to minimize the total tardiness. Let p_i, d_i denote the processing time and the due date of job i respectively. Let s_{ki} denote the setup times when job i succeeds job k immediately, where $i = 1, \ldots, n$. The tardiness of job i is denoted by T_i and T_i is defined as $T_i = \max\{C_i - d_i, 0\}$, where C_i is the completion time of job i. It is assumed that all the processing times, due dates, and setup times are non-negative integers for this problem. In addition, job preemptions are not allowed. According to the standard scheme introduced by Graham et al. (1979), the single machine scheduling problem with sequence dependent setup times minimizing the total tardiness of jobs is represented as $1|s_{ki}|\sum T_i$. Problem $1|s_{ki}|\sum T_i$ is NP-hard since its relaxation problem, problem $1||\sum T_i$, was proven to be NP-hard by Du and Leung (1990). Since the problem is NP-hard, it is unlikely that any algorithm will always find an optimal solution within polynomial computation times.

Raman et al. (1989) provided a modification of the apparent tardiness cost (ATC) heuristic rule in order to consider setup times. Lee et al. (1997) proposed a generalization of the ATC rule, called the apparent tardiness cost with setups (ATCS) rule. For the ATCS rule, the priority index was computed using the following formula,

$$I_i(t,l) = \frac{w_i}{p_i} \exp\left(-\frac{\max\{d_i - p_i - t, 0\}}{k_1 \overline{p}} \right) \exp\left(\frac{s_{li}}{k_2 \overline{s}} \right),$$

where t denotes the current time, l is the index of the job just completed, w_i is the weight of tardiness for J_i, $\overline{p}$ is the average processing time of the remaining jobs, $\overline{s}$ is the average setup time of the remaining jobs respectively, and k_1 and k_2 are two scaling parameters. It was claimed by Lee et al. (1997) that the ATCS rule is superior to the modified ATC rule proposed by Raman et al. (1989).

Ragatz (1993) presented a branch-and-bound algorithm to minimize the total tardiness with sequence dependent setup times. The lower bounds used by them are weak and the performance of the procedure is highly dependent on the problem's characteristics. A genetic algorithm was developed by Rubin and Ragatz (1995), in which a local search method based job exchange or single random exchange was embedded to improve the solutions produced by the genetic operators further. The performance of their genetic algorithm was compared to the branch-and-bound algorithm and it is claimed that their genetic algorithm is as competitive as the branch-and-bound algorithm. Tan and Narasimhan (1997) applied simulated annealing technique to minimize the total tardiness on a single machine with sequence dependent setup times. A comparison work of four algorithms, namely branch-and-bound, genetic algorithm, simulated annealing, and random start pairwise interchange was conducted by Tan et al. (2000). Based on their computational experiments, the authors concluded that the simulated annealing and random start pairwise interchange algorithms yielded good solutions within practical computation time limits. França et al. (2001) proposed a memetic algorithm (MA), in which a new solution is generated from the genetic operators of selection, crossover and mutation. A local search method is then applied to the new solution to improve it further.

Gagné et al. (2002) presented an ant colony optimization (ACO) algorithm for the single machine scheduling problem with sequence dependent setup times. The ACO algorithm utilizes a lookahead feature when selecting the next job to be included in the partial schedule. Two local search methods were included in the ACO algorithm. The first of these two methods is a restricted 3-opt method developed by Dorigo and Gambardella (1997) for the traveling salesman problem (TSP). The second method, proposed by Rubin and Ragatz (1995), is called random search pairwise interchange, which proceeds to invert pairs of adjacent jobs in turn. The ACO algorithm was compared to the branch-and-bound algorithm proposed by Ragatz (1993), the genetic algorithm and the random search pairwise interchange local search method developed by Rubin and Ragatz (1995), and the simulated annealing algorithm proposed by Tan et al. (2000). It was shown that the ACO algorithm is competitive to all four algorithms compared.

Gupta and Smith (2006) presented two algorithms, a problem space-based local search heuristic and a GRASP algorithm with path relinking for SMSPs with sequence dependent setup times. The authors claimed that the space-based local search method performs as well as the ACO algorithm and that the GRASP gave better solutions than the ACO in general. Several variants of the GRASP based algorithm, incorporating memory-based mechanisms, were proposed by Armentano and de Araujo (2006). Two mechanisms utilizing long-term memory composed from the elite solutions were applied. The first mechanism is used to influence the construction of the initial solution by extracting attributes from the elite solutions. The second mechanism makes use of path relinking to look for a better solution based on the elite solutions. The computational experiments conducted by the authors showed that their GRASP algorithms are robust and competitive to the ACO and the MA algorithms.

2.3.2 Flow Shop Scheduling Problem with Transportation Times

Under the permutation assumption of the two-machine flow shop problem, Maggu and Das (1980) solved the problem in O(*n*log*n*) by extending Johnson's (1954) algorithm for the problem $F_2 \| C_{\max}$. A Tabu search algorithm was presented by Dell'Amico (1996) for a two-machine JSSP and a FSSP with transportation times. Strusevich (1999) presented a 1.5-approximate algorithm for the two-machine open shop problem with job-dependent transportation times. It has been proved by Rayward-Smith and Rebaine (1992) that the two-machine open routing shop scheduling problem is NP-hard even when all the transportation times are equal. Lee and Strusevich (2005) presented a heuristic for both the two-machine open shop and the flow shop scheduling problems with job dependent transportation times.

2.3.3 Open Shop Scheduling Problem with Transportation Times

Strusevich (1999) considered a two-machine OSSP with transportation times to minimize the makespan. The author assumed a known time lag between the completion of an operation and the beginning of the next operation for the same job. A 1.5-approximate algorithm was proposed by the author. A $\frac{6}{5}$-approximate algorithm was proposed by Averbakh et al. (2005) for a ROSSP with two machines on a 2-node network. Averbakh et al. (2006) proved that the ROSSP with two machines on a 2-node network with *n* jobs is NP-hard; a heuristic was also developed for this problem. The same authors also proposed a heuristic for the general ROSSP based on the conclusions obtained for the routing flow shop problem.

2.4 Limitations of Prior Research Work

Based on the literature review, it is shown that only limited research work has been done on routing shop scheduling problems which consider transportation or setup times. For the single machine scheduling problem with different release dates, which is relaxed from the single machine scheduling problems with sequence dependent setup times by ignoring the transportation or setup times, there is only limited research work focusing on local search algorithms and only two exact algorithms were proposed in the prior work. Although there is some research done on single machine scheduling problems with sequence dependent setup times, as presented in Section 2.3.1, the previous work focused on problem specific algorithms which were developed for solving only one type of problem. There is even less research work reported for multiple machine routing scheduling problems. Moreover, for multiple machine routing shop scheduling problems, only 2-machine problems were analyzed and only simple heuristics were developed.

In this research, we first work on simple single machine scheduling problem. Global dominance rules and lower bound computational methods are proposed for the single machine scheduling problem with different release dates. Then we focus on more complex single machine scheduling problem by considering transportation/setup times. A generic algorithm, which is applicable to the single machine problem with/without transportation/setup times, is developed and tested based on different categories of single machine problems. At last, we presented algorithms for multiple machine routing scheduling problems and the algorithms were tested based on both benchmark problem instances and randomly generated problem instances.

Chapter 3 Branch-and-Bound Algorithm for Solving Single Machine Total Weighted Tardiness Problem with Unequal Release Dates

In this chapter, we present a branch-and-bound algorithm for solving the single machine total weighted tardiness problem. The objective of the problem is to schedule the jobs to minimize the total weighted tardiness. Three global dominance rules are proposed to reduce the search tree, and a method to compute the lower bound of the total weighted tardiness is also introduced. The resulting branch-and-bound algorithm has been implemented and the computational results show that the dominance rules are efficient in reducing the size of the search tree and computation time. The computational experiments conducted in this research also provide useful guidelines for future implementation of the branch-and-bound algorithm to solve the single machine total weighted tardiness problem.

3.1 Introduction

This chapter considers a single machine scheduling problem with the objective of minimizing total weighted tardiness. Akturk and Ozdemir (2000) states the problem as follows. For a single machine, n independent jobs are released continuously, where each job has a processing time p_i, a release date r_i, a due date d_i, and a tardiness penalty weight w_i. Minimizing the total weighted tardiness is one of the important objectives for the single machine scheduling problem in practice. In manufacturing environments, different orders have different priorities and accordingly, different penalties for delayed deliveries. By minimizing the total weighted tardiness, unacceptably long waiting times of any given job is likely to be avoided, especially for those jobs with high priority. Besides manufacturing environments, minimizing total weighted tardiness is also important in a multi-tasking computer operating system,

where one of the functions of the system is to perform scheduling such that the central processing unit (CPU) can be devoted to different programs or processes in order to complete all tasks without any long delays.

According to the standard scheme introduced by Graham et al. (1979), the single machine total weighted tardiness problem is represented as $1|r_i|\sum w_i T_i$. This problem is known to be NP-hard in the strong sense (Jouglet et al. 2004) and even its relaxed problem, problem $1||\sum w_i T_i$, is proven to be NP-hard by Lenstra et al. (1977). While many types of single machine scheduling problems have been well studied by researchers, few have focused on problem $1|r_i|\sum w_i T_i$. Akturk and Ozdemir (2000) developed the first branch-and-bound approach for problem $1|r_j|\sum w_j T_j$. However, the lower bound computational method proposed by Akturk and Ozdemir (2000) ignores job releasing date difference and is very weak. Moreover, the dominance rules proposed by the authors are condition-based rules, whose validity is based on partial schedules. A constraint programming based branch-and-bound algorithm was proposed by Jouglet et al. (2004). Their branch-and-bound algorithm focuses on searching strategies and valid lower bound computational method was not discussed.

In this chapter, we present an efficient branch-and-bound algorithm to solve problem $1|r_i|\sum w_i T_i$. Three global dominance rules are developed to eliminate non-optimal schedules. Any schedule violating the dominance rules can be removed from searching candidates. Moreover, a lower bound computational method, which considers job release dates, is also developed in this research work.

The following symbols are used throughout this chapter.

J_i	Job *i*, where *i* is the index of a job
N	The number of jobs in a single machine scheduling problem

p_i	The processing time of J_i
r_i	The release date of J_i
d_i	The due date of J_i
w_i	The tardiness penalty weight for J_i
C_{ij}	The lower bound of completion time if J_i is scheduled at the j^{th} position
T_{ij}	The lower bound of weighted tardiness of J_i if it is scheduled at the j^{th} position
A_i	The set of jobs that have to be processed after J_i according to the dominance rules
B_i	The set of jobs that have to be processed before J_i according to the dominance rules
S	A set of jobs
$\|S\|$	The number of elements in set S
Sum_S	The sum of processing time for jobs in set S
LBC_S	The lower bound of completion time for jobs in set S
UBC_S	The upper bound of optimal schedule completion time for jobs in set S
LB	The lower bound of total weighted tardiness for the problem
LUB	The local upper bound schedule of the unscheduled jobs
UB	The upper bound schedule of the problem, which is the current best schedule
T(Schedule)	The total weighted tardiness of a schedule or partial schedule
$\lceil x \rceil$	The smallest integer greater than or equal to x

This chapter is organized as follows. In Section 3.2, three global dominance rules and a local dominance rule are presented. Section 3.3 introduces a lower bound computational method for problem $1|r_i|\sum w_iT_i$ by formulating and solving appropriate assignment problems. The details of the implementation of the branch-and-bound algorithm are given in Section 3.4. The proposed algorithm has been coded in C++. The computational results of applying the proposed algorithm to certain problem instances are reported in Section 3.5, and conclusions as well as some possible future research work are provided in Section 3.6.

3.2 Dominance Rules

In this section, we present three global dominance rules for problem $1|r_i|\sum w_iT_i$. The three global dominance rules are based on the dominance rules

developed by Emmons (1969) for problem $1||\sum T_i$ and the dominance rules developed by Rinnooy Kan et al. (1975) for problem $1||\sum w_i T_i$. We first define some concepts that will be used to present the dominance rules. In an optimal schedule, B_i is used to denote the set of jobs preceding J_i and A_i is used to denote the set of jobs following J_i. For a set of jobs S with $|S| = n$, we can schedule the jobs in S in nondecreasing order of the release dates with time complexity O($n$log$n$) and the makespan obtained is the lower bound of the optimal schedule completion time, denoted by LBC_S. The upper bound of the optimal schedule completion time for the set of jobs in S can be obtained with time complexity O(n) from the function below,

$$UBC_S = \max_{i \in S}\{r_i\} + \sum_{i \in S} p_i \ .$$

It is noted that UBC_S can possibly be reduced further if some of the global dominance relationships are known and considered.

The three global dominance rules are presented below and the proof is given in Appendix A.

Global dominance rule 1A: Let J_i and J_k be two jobs ($i, k \in S$). If

(a) $r_i \le r_k$,

(b) $w_i \ge w_k$,

(c) $p_i = p_k$, and

(d) $d_i \le \max\left\{d_k, \max\left\{r_k, LBC_{B_k}\right\} + p_k\right\}$,

then J_i precedes J_k.

Rule 1A ensures that total weighted tardiness will not increase when jobs J_i and J_k are exchanged. Condition (d) in Rule 1A is to make sure that the total tardiness of jobs J_i and J_k will not increase when the due date of job J_i is not later than that of job J_k or the earliest completion time of job J_k, which is given by $\max\left\{r_k, LBC_{B_k}\right\} + p_k$.

Condition (a) in Rule 1A guarantees that exchanging jobs will not violate release date constraints, as shown in Figure 3.1. Condition (b) in Rule 1A ensures that the total weighted tardiness of jobs J_i and J_k will not increase after the exchange. Condition (c) in Rule 1A can guarantee that the start time of job J_k after the exchange is the same with that of jobs J_i before the exchange and therefore the start time of the other jobs are not affected. Therefore, Rule 1A is able to guarantee that the total weighted tardiness for the schedule will not increase after the exchange.

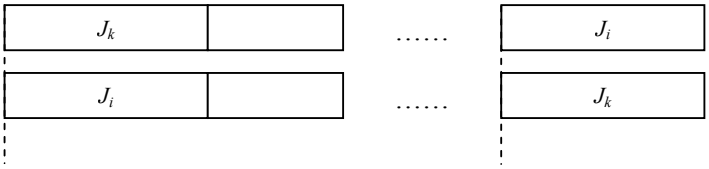


Figure 3.1 Illustration of exchanging jobs

If all the jobs have the same release date, the following global dominance rule proposed by Rinnooy Kan et al. (1975) is valid. Condition (a) in Rule 1A can be removed and Condition (c) in Rule 1A can be relaxed in the following Rule 1 (b) when the release dates of all jobs are the same.

Global dominance rule 1B: Let J_i and J_k be two jobs ($i, k \in S$). If

(a) $w_i \geq w_k$,

(b) $p_i \leq p_k$, and

(c) $d_i \leq \max\{d_k, Sum_{B_k} + p_k\}$,

then J_i precedes J_k.

Global dominance rule 1A is dominated by global dominance rule 1B due to the relaxation of conditions (a) and (b) in Global dominance rule 1A. Therefore, it is preferable to apply global dominance rule 1B instead of global dominance rule 1A when all the release dates of the unscheduled jobs are the same.

Global dominance rule 2: Let J_i and J_k be two jobs ($i, k \in S$). If

(a) $r_i \le r_k$,

(b) $p_i = p_k$, and

(c) $d_k \ge UBC_S - Sum_{A_i}$, then J_i precedes J_k.

In Global dominance rule 2, the expression $UBC_S - Sum_{A_i}$ in Condition (c) is the latest completion time of job J_i in any optimal schedule. As $d_k \ge UBC_S - Sum_{A_i}$, job J_k will not be tardy after it is exchanged with job J_i. Therefore, there exists at least one optimal solution with J_i preceding J_k when the conditions in Global dominance rule 2 are satisfied.

Global dominance rule 3: For any job J_k ($k \in S$), if $d_k \ge UBC_S$, then J_k can be assigned last. In the situation that there are $m \ge 1$ jobs satisfying $d_k \ge UBC_S$, then the m jobs can be assigned in the last m positions in any sequence without sacrificing the optimality of the schedule.

It is noted that there always exists at least an optimal schedule if global dominance rules are followed.

Local dominance rule

Local dominance rules were originally presented by Akturk and Ozdemir (2001) and Jouglet et al. (2004, 2008) for the single machine scheduling problem with different release dates. We have implemented a simplified version of these local dominance rules: Let J_i and J_k be two jobs scheduled next to each other in a schedule or partial schedule. If interchanging the positions of J_i and J_k can produce a schedule which satisfies

(1) same completion time with smaller weighted tardiness, or

(2) smaller completion time with equal or smaller weighted tardiness,

then we can say that the current schedule is not locally optimal and the total weighted tardiness will decrease or not change if the positions of the two jobs are interchanged. Here, we use local dominance both to prune the search tree and to explore the neighborhood.

In our proposed branch-and-bound algorithm, a local search method applying the above local dominance rule is developed to improve the upper bound solution obtained. The implementation of the local search method is based on a backtracking strategy. Whenever an improvement is made by interchanging two adjacent jobs according to the above two conditions, the search procedure will backtrack one position to check for possible improvement. This backtracking search procedure is repeated until no improvement is possible. It is noted that the final schedule obtained by the above local search method may not be adjacent pairwise interchange optimal as another condition, namely the completion time increases but the total weighted tardiness decreases for two adjacent jobs, is not considered. This is to reduce computation time that is incurred to compute the change of the total weighted tardiness. Another reason is that it is impossible to determine whether a partial schedule is locally optimal or not when some jobs have not been scheduled.

As mentioned by Rinnooy Kan et al. (1975), the implementation of the global dominance rules may generate a precedence cycle when two rules contradict each other. To overcome this problem, the global dominance rules are implemented based on the procedure proposed by Rinnooy Kan et al. To avoid precedence cycles, only the pair of jobs (i, k) without any relationship is considered to find a possible precedence sequence by applying the three global dominance rules. Whenever a new precedence J_i preceding J_k is found, the transitive closure of the set of known precedence

relationships will be constructed immediately. For example, if we find that J_i precedes J_k based on any of the global dominance rules, then J_k and all the jobs following J_k (exclude J_i itself) will follow J_i, and J_i and all the jobs preceding J_i will precede J_k as illustrated in Figure 3.2. Thus, the sets of jobs, $A_j \coloneqq A_j \bigcup \{k\} \bigcup A_k$ for every $j \in \{i\} \bigcup B_i$, and the sets of jobs $B_l \coloneqq B_l \bigcup \{i\} \bigcup B_i$ for every $l \in \{k\} \bigcup A_k$, are constructed accordingly. Formula $A_j \coloneqq A_j \bigcup \{k\} \bigcup A_k$ for every $j \in \{i\} \bigcup B_i$ denotes that if job J_i is scheduled before job J_k, job J_k and its following jobs will be scheduled after all those jobs which are scheduled before job J_i. Similarly, formula $B_l \coloneqq B_l \bigcup \{i\} \bigcup B_i$ for every $l \in \{k\} \bigcup A_k$ means that if job J_i is scheduled before job J_k, all jobs which precede job J_i will be preceding jobs of job J_k and its following jobs.

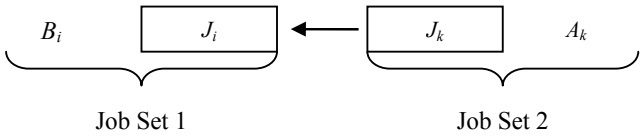


Figure 3.2 Job relationships after exchanging jobs

3.3 Lower Bound

For problem $1|r_i|\sum w_iT_i$, relaxing the release date or weight or both will produce problems $1||\sum w_iT_i$, $1|r_i|\sum T_i$ and $1||\sum T_i$ respectively, which are also NP-hard problems. Akturk and Ozdemir (2000) used the lower bound developed for $1||\sum w_iT_i$ to compute the lower bound for problem $1|r_i|\sum w_iT_i$. Jouglet et al. (2004) computed the lower bound for problem $1|r_i|\sum w_iT_i$ based on the job splitting lower bound computational method developed by Belouadah et al. (1992) for problem $1|r_i|\sum w_iC_i$. In this study, we extend the method proposed by Rinnooy Kan et al. (1975), in which the lower bound is computed by solving an appropriate assignment

problem. The cost coefficients of the assignment problem are the lower bounds of the weighted tardiness by putting job J_i at the j^{th} position in the schedule.

In order to consider arbitrary release dates, the formulae defined by Rinnooy Kan et al. (1975) are modified to compute the lower bound for problem $1|r_i|\sum w_i T_i$. The modified formulae are given below:

$$R_i(K) = \{q \in Q \mid Q \subseteq \{S - (B_i \cup \{J_i\} \cup A_i)\}, |Q| = K\},$$

$$C_{ij} = LBC_{B_i \cup J_i \cup R_i(j-|B_i|-1)},$$

$$T_{ij} = \begin{cases} \max\{C_{ij} - d_i, 0\} \times w_i & \text{for } |B_i| < j \le |\{1,...,n\} - A_i| \\ +\infty & \text{otherwise} \end{cases}, \qquad (3.1)$$

where $R_i(K)$ denotes choosing K jobs from a given set of jobs and S denotes the set of unscheduled jobs. The expression $j-|B_i|-1$ is the number of jobs to be scheduled before job J_i except for jobs in B_i and $C_{ij} = LBC_{B_i \cup J_i \cup R_i(j-|B_i|-1)}$ is the lower bound of completion time of job J_i if it is scheduled at j^{th} position. T_{ij} is the lower bound of weighted tardiness of job J_i. As pointed in Baker (1974), an optimal schedule can easily be obtained by a polynomial time algorithm where jobs are scheduled in order of nondecreasing release dates for problem $1|r_j|C_{\max}$. From the definition of $R_i(K)$, we know that the number of sets Q increases exponentially with the increase of $|S - (B_i \cup \{J_i\} \cup A_i)|$, and hence it is difficult to find the set Q which minimizes C_{ij}. However, we can get the lower bound of completion time for K jobs by scheduling all jobs in set $S - (B_i \cup \{J_i\} \cup A_i)$ in nondecreasing order of release dates and then replace the processing times of the first K jobs with the K smallest processing times.

Here, we construct K virtual jobs in order to design a polynomial method to compute C_{ij}. The steps of constructing virtual jobs are given below:

Step 1: Let Q_1 denote the jobs from the set $S-(B_i \cup \{J_i\} \cup A_i)$ with the smallest release date, where $|Q_1| = K$ and the jobs in Q_1 are sorted in nondecreasing order of release date;

Step 2: Let Q_2 denote the jobs from the set $S-(B_i \cup \{J_i\} \cup A_i)$ with the smallest processing time, where $|Q_2| = K$ and the jobs in Q_2 are sorted in nondecreasing order of processing time;

Step 3: Construct K new jobs as follows: The i^{th} new job is constructed by using the i^{th} smallest release date in Q_1 as the new job's release date and the i^{th} smallest processing time in Q_2 as the new job's processing time.

Then C_{ij} can be obtained by scheduling jobs $B_i \cup R_i(k)$ in nondecreasing order of release date and followed by job J_i. Based on the procedure described above, the time complexity for computing C_{ij} is $O(|S-(B_i \cup \{J_i\} \cup A_i)| \log(|S-(B_i \cup \{J_i\} \cup A_i)|))$ when A_i and B_i are known.

By defining $x_{ij} = 1$ if job J_i is scheduled at the j^{th} position and 0 otherwise, we can formulate the mathematical programming model of the assignment problem to compute the lower bound of weighted tardiness as follows:

$$\min \quad \sum_{i=1}^{n}\sum_{j=1}^{n} T_{ij}x_{ij},$$

$$\sum_{j=1}^{n} x_{ij} = 1 \text{ for } i = 1, ..., n,$$

$$\sum_{i=1}^{n} x_{ij} = 1 \text{ for } j = 1, ..., n,$$

$$x_{ij} \geq 0 \text{ for } i, j = 1, .., n. \qquad (3.2)$$

Note that the dual problem of the above assignment problem is given by

$$\max \quad \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j \,,$$

$$u_i + v_j \le T_{ij} \text{ for } i, j = 1, ..., n \,. \qquad (3.3)$$

We assume that an optimal solution to the assignment problem (x_{ij}^*) has the objective value LB^* and a corresponding dual solution (u_i^*, v_j^*). As C_{ij} will not decrease with the increase of j, T_{ij} will not decrease either. If the lower bound for the descendant assignment problems after scheduling job J_k at the l^{th} position is LB^*, then (u_i^*, v_j^*), where $i \neq k$ and $j \neq l$, is a feasible solution to the dual problem of the descendant node of the search tree. Therefore, the lower bound for the descendant nodes can be obtained with time complexity $O(1)$ by the formula given below:

$$\sum_{i \neq k} u_i^* + \sum_{j \neq l} v_j^* = LB^* - u_k^* - v_l^* \,. \qquad (3.4)$$

It is easy to see that the lower bound obtained by Eq. (3.1) is stronger than that obtained by Eq. (3.4).

Our preliminary computational experiments showed that the branch-and-bound algorithm spent most of the time on computing the lower bound solution by solving the assignment problem. There are two methods to reduce this computation time:

(1) Use the dual solution of the parent node as the initial dual solution of the descendant assignment problem to reduce the computation effort;

(2) Do not solve the assignment problem at every node as Eq. (3.4) can be used to get a lower bound for each of the descendant nodes.

From our experiments, we found that the first method was unable to reduce the computation time efficiently and so we focus on the second method. As pointed out by

Rinnooy Kan et al. (1975), it is necessary to develop a stronger lower bound especially for the upper levels of the search tree for a large reduction in the search tree. While in the deep levels of the search tree, it is preferable to use a simple lower bound combined with extensive enumeration. Based on this strategy, they proposed using the lower bounds of varying computational complexities throughout the search tree, known as the gliding lower bound. However, their computational experiments only showed a small decrease in computation time.

For the branch-and-bound algorithm proposed in this chapter, experiments will be conducted to compare the efficiency of the three strategies introduced by Rinnooy Kan et al. (1975), with the computational results being presented in Section 3.6. The three strategies are described below:

I. Find the global dominance rules and solve the assignment problem at the root node, and then use Eq. (3.4) to get the lower bounds at the descendant nodes;

II. Find the global dominance rules and solve the assignment problem at every node;

III. Find the global dominance rules and solve the assignment problem at the upper tree levels, and then use Eq. (3.4) to get the lower bounds at the deep tree levels.

Although the above three strategies are discussed in (1975), the details of implementing strategy III were not given. In order to implement strategy III, we determine the tree levels at which the global dominance rules are checked and the assignment problems are solved to get the lower bounds. These tree levels are obtained from the following set:

$$L \in \left\{ l \mid l = n - \left\lceil \sqrt{2nd - d^2} \right\rceil \right\}, \quad \text{for } d \in \{1, 2, ..., n\}.$$

It is noted that the formula $\sqrt{2nd - d^2}$ represents one quarter of the y values in a circle centered at (n, 0). As an example, let n = 10. Figure 3.3 illustrates that the set of y values is {4.4, 6.0, 7.1, 8.0, 8.7, 9.2, 9.5, 9.8, 9.9, 10.0} for the corresponding integer values of x from 1 to 10, and hence we can get the tree levels from the set {0, 1, 2, 4, 5} based on the above formula. While there could be other methods to implement strategy III, we find that our proposed procedure is simple and can be implemented easily. Moreover, there is no necessity to fine-tune any parameter for this procedure.

Based on the method presented above for strategy III, it can be seen that the lower bounds are obtained at the predetermined upper levels of the search tree by solving the relevant assignment problems and also by Eq. (3.4) at the other levels of the search tree.

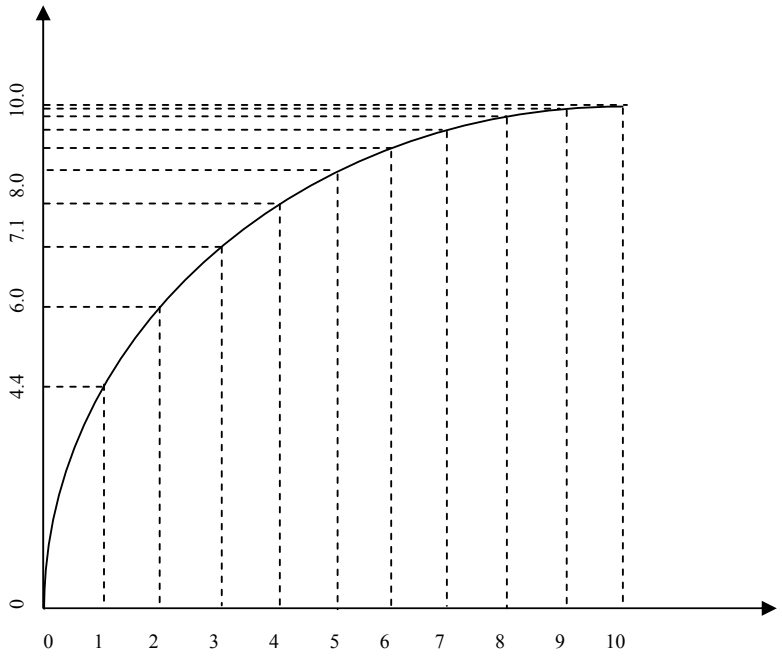


Figure 3.3 10-job problem example for Strategy III

3.4 Branch-and-Bound Procedure

For a single machine scheduling problem, any enumeration scheme is able to find and verify the optimal solution if sufficient computation time is given. However, for the NP-hard problem $1|r_i|\sum w_iT_i$, the computation time increases significantly with the size of the problem and hence it is too time consuming to perform direct enumeration search. As such, we present an efficient search tree enumeration method together with some search tree reduction criteria in this section. The details of the implementation of the branch-and-bound algorithm are also provided.

3.4.1 Enumeration Method

A simple enumeration method was proposed by Rinnooy Kan et al. (1975) for problem $1||\sum w_iT_i$. From the root node without any job being scheduled, *n* different nodes are branched from the first level with each node corresponding to a specific job being scheduled at the first position. Each of these nodes will produce $n - 1$ new nodes on the second level, corresponding to one of the remaining $n - 1$ jobs filling the second position of the schedule. The whole search tree can thus be generated by implementing this procedure.

To speed up the search, we modify the above enumeration procedure. We first present a ATC priority rule to generate an upper bound solution for problem $1|r_i|\sum w_iT_i$. It was shown by Vepsalainen and Morton (1987) that the ATC rule outperforms other priority rules in minimizing the weighted tardiness for the JSSP. Based on the ATC rule, the job assignment priority index is computed using the formula given below:

$$ATC_i(t) = \frac{w_i}{p_i} \exp\left(-\frac{\max\{d_i - p_i - t, 0\}}{k\bar{p}} \right), \tag{3.5}$$

where $\bar{p}$ is the average processing time of the unscheduled jobs, t is the current time and k is a lookahead parameter. A fixed value of $k = 1$ is used in our computational experiments because our preliminary experimentation shows that $k = 1$ can produce slightly better results than other values of k in general.

From the root node, the ATC values are computed for the n jobs and these values are sorted in nonincreasing order. Then n different nodes are branched with each job corresponding to one of the n jobs at the first level in the order of the sorted ATC values. Each of the tree nodes will be branched to generate $n - 1$ nodes for the second level and the procedure is repeated to enumerate the search tree. The purpose of using the ATC priority rule here is to try to prevent the depth-first search of the branch-and-bound algorithm from making a wrong choice and getting trapped with going down a very deep tree level when a different choice would have led to a better schedule.

3.4.2 Tree Reduction Criteria

The method described in the previous section is an explicit enumeration scheme that has to be incorporated with a search tree reduction method to make it more efficient. The following search tree reduction criteria are applied in the proposed branch-and-bound algorithm to eliminate certain search tree nodes:

1) Active schedule generation rule;
2) Global dominance relationships;
3) Local dominance rule;

4) Lower bound.

As it is well known from Baker (1974) that any optimal schedule must be an active schedule for the machine scheduling problem, the active schedule generation rule is used to prune the non-active schedules. If S is the set of unscheduled jobs, then let $C^* = \min\{C_i \mid i \in S\}$ be the earliest completion time among all the unscheduled jobs. Only those jobs that satisfy $r_i < C^*$ will be considered for scheduling at the current node in order to generate an active schedule. If a job has at least one preceding job based on the global dominance rules, the node is fathomed and hence eliminated from branching; otherwise it is possible to branch from this node. If the schedule generated after scheduling a job is not locally optimal, the node can also be pruned from branching. A lower bound (LB) of the total weighted tardiness for a single machine scheduling problem can be obtained by

$$LB = LB^* + T(S^{'}),$$

where $S^{'}$ denotes the scheduled jobs and $T(S^{'})$ denotes the total weighted tardiness for the scheduled jobs, and LB^* can be obtained based on the methods described in Section 3.4. Thus, a tree node is fathomed if LB at the current tree node is greater than or equal to the total weighted tardiness of the current best schedule, which is the upper bound (UB) schedule. The total weighted tardiness of the UB schedule is denoted by $T(UB)$.

3.4.3 Implementation of the Branch-and-Bound Algorithm

The depth-first branch-and-bound algorithm is implemented with a recursive routine that calls itself on each of its descendant nodes in turn. In the branch-and-bound routine, a local upper bound (LUB) schedule is obtained by the ATC rule for the

unscheduled jobs. We use $T(LUB)$ to denote the total weighted tardiness of the *LUB* schedule. The main search routine and the branch-and-bound routine are given below:

Solver( )

```
BEGIN
    Obtain an initial UB schedule by the ATC rule;
    Find job dominance relationships;
    Compute the cost matrix values for the assignment problem;
    Solve the assignment problem to obtain LB* and (u_i*, v_j*);
    IF (LB* < T(UB)) THEN
        Branch_and_Bound (unscheduled jobs);
    ELSE
        UB schedule is optimal, stop;
    ENDIF
END
```

Branch_and_Bound (unscheduled jobs)

```
BEGIN
    IF (partial schedule is local optimal) THEN
        IF (number of unscheduled jobs > 0) THEN
            IF (construct assignment problem condition satisfied) THEN
                Find job dominance relationships for the unscheduled jobs;
                Compute the cost matrix values for the assignment problem;
                Solve the assignment problem to obtain LB* and (u_i*, v_j*);
            ELSE
                LB* := LB* - u_k* - v_l*;  /* Here k and l refer to job J_k being */
                    /* scheduled in the l^th position at one */
                                /* upper level of the current level */
            ENDIF
            Get LUB schedule by the ATC rule;
            IF (T(S') + T(LUB) < T(UB)) THEN
                Replace UB schedule by a combination of the partial schedule
                and LUB schedule;
            ENDIF
            IF (LB* + T(S') < T(UB)) THEN
                Apply tree reduction criteria;
                FOR EACH branching node
                    Schedule a job;
                    Branch_and_Bound (remaining unscheduled jobs);
                END FOR
            ENDIF
        ENDIF
    ENDIF
END
```

3.5 Computational Results

To test the performance of the proposed branch-and-bound algorithm, random problem instances are generated with a scheme similar to that in Akturk and Ozdemir (2001). As pointed out in Akturk and Ozdemir (2000) it is difficult for commercial optimization software packages, such as ILOG CPLEX, to find an optimal solution even for a 10-job problem. Therefore, the sizes of the problems in the computational experiments carried out are limited to 10, 20 and 30 jobs. Each instance is generated based on four uniformly distributed parameters r_i, p_i, d_i and w_i. The values of p_i and w_i are uniformly distributed on some bounded interval. The distributions of r_i and d_i depend on two parameters: α and β. For each job J_i, r_i is generated from the uniform distribution on $[0, \alpha\sum_i p_i]$ and $d_i - (r_i + p_i)$ is generated from the uniform distribution on $[0, \beta\sum_i p_i]$, where $\alpha \in \{0, 0.5, 1, 1.5\}$ and $\beta \in \{0.05, 0.25, 0.5\}$. The settings for generating the random problem instances are given in Table 3.1 and 10 random instances are generated for each combination of the settings.

Table 3.1 Settings for generating problem instances

Factor	Setting
Number of jobs	10, 20, 30
Variability of p_i	[1, 10]
Variability of w_i	[1, 10]

The branch-and-bound algorithm has been coded in C++ and the assignment problem is solved by ILOG CPLEX Network Simplex algorithm (2006). One of the most efficient implementation of the Network Simplex algorithm reported in literature is by Goldberg et al. (1989) with a complexity of $O(n^3 \log n)$. The computational experiments are carried out on a Pentium IV PC with 2.6GHz CPU and 512MB RAM

running on Windows XP operating system. All the computation times reported are in seconds.

3.5.1 Computational Comparison of Lower Bounds

To evaluate the efficiency of the lower bound computational method proposed in this study, computational experiments are carried out to compare the lower bound used by Jouglet et al. (2004), denoted as LB_1, and the assignment problem-based lower bound proposed in this chapter, denoted as LB_2. For $LB_{1,}$ the lower bound of the weighted completion time is first computed based on the general job splitting method described in Belouadah et al. (1992). Then the lower bound of the weighted tardiness is computed by $LB_1 = LB_{\sum w_i C_i} - \sum w_i d_i$, where $LB_{\sum w_i C_i}$ is the lower bound of the weighted completion time obtained by general job splitting method.

Our preliminary computational results show that the total weighted tardiness for the problem instances approaches zero when $\beta > 0.5$. Therefore, our computational comparison of lower bound values is only based on $\beta \in \{0.05, 0.25, 0.5\}$. For each combination of n, α and β, 10 problem instances are generated randomly. The minimum lower bound, the average lower bound and the maximum lower bound among the 10 random instances are given in Table 3.2. The larger average lower bounds obtained by the two lower bound computation methods are highlighted using bold font. Since the tightness of due dates is determined by β, it is expected that LB_2 would give a tighter lower bound than LB_1 when β is large because the due dates are ignored when LB_1 is computed. This is reflected by the computational results presented in Table 3.2, which indicate that the average lower bound value of LB_2

tends to exceed LB_1 when β increases from 0.05 to 0.5. When $\beta > 0.5$, the tightness of LB_1 and LB_2 is the same because the optimal value of the total weighted tardiness is 0.

Table 3.2 Comparison of lower bounds

n	α	β	LB_1			LB_2		
			Min.	Avg.	Max.	Min.	Avg.	Max.
10	0.0	0.05	503	**764.9**	1452	473	683.9	1405
		0.25	306	**521.5**	836	258	490.9	852
		0.5	0	71.3	285	67	**183.9**	271
	0.5	0.05	60	**280.6**	521	72	219.1	508
		0.25	0	85.6	313	53	**155.3**	275
		0.5	0	0.0	0	0	**29.4**	80
	1.0	0.05	26	**151.6**	308	1	80.0	275
		0.25	0	0.0	0	0	**15.6**	65
		0.5	0	0.0	0	0	**3.1**	31
	1.5	0.05	0	**12.7**	64	0	7.1	17
		0.25	0	0.0	0	0	0.0	0
		0.5	0	0.0	0	0	**3.8**	30
15	0.0	0.05	1213	**1807.2**	2191	1109	1629.3	1997
		0.25	616	**1229.8**	1973	530	1017.1	1902
		0.5	0	364.1	983	169	**550.4**	1008
	0.5	0.05	362	**697.0**	971	278	410.2	704
		0.25	0	75.3	276	95	**185.2**	319
		0.5	0	0.0	0	0	**62.5**	212
	1.0	0.05	0	**159.5**	414	0	33.3	119
		0.25	0	0.0	0	0	**10.2**	67
		0.5	0	0.0	0	0	0.0	0
	1.5	0.05	0	**33.3**	198	0	9.3	63
		0.25	0	0.0	0	0	0.0	0
		0.5	0	0.0	0	0	0.0	0
20	0.0	0.05	2262	**3110.0**	3803	1748	2639.2	3253
		0.25	1494	**2110.9**	3205	1370	1771.0	2929
		0.5	0	603.0	1692	537	**800.0**	1345
	0.5	0.05	632	**1205.2**	1776	403	822.6	1358
		0.25	0	213.6	1081	182	**386.0**	759
		0.5	0	0.0	0	0	**158.6**	423
	1.0	0.05	69	**304.6**	859	0	98.2	299
		0.25	0	0.0	0	0	**4.2**	15
		0.5	0	0.0	0	0	0.0	0
	1.5	0.05	0	**19.9**	117	0	0.8	4
		0.25	0	0.0	0	0	0.0	0
		0.5	0	0.0	0	0	0.0	0

3.5.2 Efficiency of Dominance Rules

To evaluate the performance of the global dominance rules, the number of dominance relationships found by the three global dominance rules is summarized in Table 3.3 for problem instances with different characteristics. The preliminary computational experiments show that the efficiency of the global dominance rules depends on problem characteristics. Moreover, it is also shown that none of the three global dominance rules always dominates any other dominance rules. For a given problem instances with unknown problem characteristics, it is preferred to apply all the three dominance rules to prune more search nodes. The minimum number of dominance relationships, the average number of dominance relationships and the maximum number of dominance relationships of 10 randomly generated problem instances for different combinations of n, α and β are given in the last three columns of Table 3.3 respectively. It can be seen from Table 3.3 that the average number of dominance relationships tends to decrease with the increase of α. When $\alpha > 0$, the range of release dates increases with the increase of α and hence the chance of satisfying the condition $r_i \leq r_k$ becomes lower. When $\alpha = 0$, all the jobs have the same release times and hence the global dominance rule 1B, which dominates global dominance rule 1A, is valid and could find more dominance relationships than global dominance rule 1A.

In general, the average number of dominance relationships decreases slightly with the increase of β. When the value of β is small, the range of the due dates is small and hence the global dominance rule 1A is likely to be valid because its condition (d) has a large possibility of being satisfied. However, when the value of β is large, the range of the due dates is large and hence the global dominance rules 2 and

3 are likely to be valid. It is noted that the global dominance rules 1A, 2 and 3 are valid only when the processing times of the two jobs are the same. Therefore, it is possible that less dominance relationships can be found when the range of the processing times is large.

Table 3.3 Global dominance relationships

n	α	β	Number of global dominance relationships found		
			Min.	Avg.	Max.
10	0.0	0.05	16	24.5	31
		0.25	11	16.7	23
		0.5	11	18.5	32
	0.5	0.05	0	3.1	5
		0.25	1	2.6	4
		0.5	0	2.1	4
	1.0	0.05	1	2	4
		0.25	0	1.9	3
		0.5	0	2.2	5
	1.5	0.05	1	3.4	6
		0.25	0	2.1	4
		0.5	1	2.7	5
15	0.0	0.05	42	61.1	79
		0.25	24	35.2	44
		0.5	10	31.3	46
	0.5	0.05	2	5	8
		0.25	2	5.7	12
		0.5	1	5.3	10
	1.0	0.05	4	7.1	10
		0.25	3	4.9	7
		0.5	1	4.7	8
	1.5	0.05	3	7.3	11
		0.25	1	5.3	11
		0.5	3	5.3	10
20	0.0	0.05	70	95.7	113
		0.25	42	68.1	79
		0.5	44	67.5	88
	0.5	0.05	6	12.2	26
		0.25	4	10	15
		0.5	2	6.3	11
	1.0	0.05	3	10.9	21
		0.25	5	9	19
		0.5	5	10.9	15
	1.5	0.05	3	9.8	19
		0.25	4	9.7	16
		0.5	0	8.5	15

To further test the efficiency of the global and local dominance rules, additional computational experiments are carried out. When the size of the problem is small, the optimal solution can be found and verified within a very short computation time, when the size of the problem is large, only a few problems can be solved optimally. Therefore, we chose n = 20 jobs based on our preliminary computational experiments to test the efficiency of the dominance rules in reducing the number of nodes of the search tree and the computation time. As pointed out in Chu (1992) for problem $1|r_i|\sum T_i$, there is no simple relationship between problem difficulty and the variation of β, except when β is very large. In this case, the due dates are so scattered such that there are many solutions with zero tardiness. When α is small, the problem difficulty increases with β.

In our computational experiments, the branch-and-bound algorithm has been run with n = 20 jobs using various combinations of α and β. For each lower bound implementation strategy described in Section 3.4, the branch-and-bound algorithm has been run under the following conditions, (a) without global and local dominance rules, (b) with only the local dominance rule, and (c) with both global and local dominance rules. Our preliminary computational experiments show that there is no interaction between the local dominance rule and the global dominance rules. Therefore, we do not present the computational results for the global dominances without local dominance rule. The computational experiments are carried out with a computation time limit of 600 seconds for each problem instance as our preliminary computational experiments show that schedule quality improvement is marginal even though computation time is extended to 3600 seconds. It is pointed out here that setting a time limit does not mean transforming an exact algorithm to a heuristic one, but is helpful to

evaluate the performance of the exact algorithms when solving difficult problem instances.

Table 3.4 Comparison of efficiency of dominance rules based on Strategy I

α	β	(a)			(b)			(c)		
		#opt.[1]	#nodes[2]	Time[3]	#opt.	#nodes	Time	#opt.	#nodes	Time
0.0	0.05	10	550	0.19	10	357	0.12	10	23	0.04
	0.25	0	3,624,509	600.01	5	2,254,387	465.84	10	14,695	3.42
	0.5	0	5,331,096	600.01	0	3,915,386	600.01	7	1,796,872	300.10
0.5	0.05	0	4,778,997	600.01	8	1,437,915	241.01	10	306,720	55.82
	0.25	0	6,405,399	600.01	0	4,131,127	600.01	4	2,845,301	417.57
	0.5	0	7,571,075	600.01	0	4,839,537	600.01	0	4,800,565	600.01
1.0	0.05	5	3,726,173	328.46	10	164,079	20.94	10	32,291	4.47
	0.25	6	4,221,314	290.25	10	240,029	24.62	10	122,478	12.65
	0.5	6	3,751,128	246.95	7	1,736,099	184.79	9	989,015	114.83
1.5	0.05	10	42,198	5.45	10	842	0.16	10	356	0.07
	0.25	10	28,166	2.51	10	428	0.07	10	264	0.05
	0.5	9	567,447	60.01	10	329,116	51.15	10	25,589	3.76

1 – Number of optimal solutions found and verified within 600 seconds among the 10 instances
2 – The average number of nodes explored
3 – The average computation time for the 10 random instances

The computational results are summarized in Tables 3.4, 3.5 and 3.6. Table 3.4 indicates that when the dominance rules are applied, the total number of optimal solutions which are found and verified within 600 seconds increases from 56 for setting (a) to 80 for setting (b), and then to 100 for setting (c). The average number of nodes explored and the average computation time decrease significantly when more dominance rules are applied, as shown in the "#nodes" and "Time" columns respectively. The computational results show that the local dominance and global dominance rules are efficient in reducing the search tree when they are applied in the branch-and-bound algorithm. Tables 3.5 and 3.6 also show similar results as Table 3.4. Hence we can conclude that the dominance rules proposed in this chapter are efficient in reducing the size of the search tree and are independent of the lower bound strategy.

Table 3.5 Comparison of efficiency of dominance rules based on Strategy II

α	β	(a)			(b)			(c)		
		#opt.	#nodes	Time	#opt.	#nodes	Time	#opt.	#nodes	Time
0.0	0.05	10	534	0.25	10	347	0.24	10	22	0.04
	0.25	0	1,609,093	600.01	5	682,489	449.84	10	13,054	12.08
	0.5	0	1,472,419	600.01	3	698,939	512.68	7	416,321	319.62
0.5	0.05	5	937,252	423.05	10	44,663	33.82	10	17,690	14.06
	0.25	3	1,262,324	489.67	8	315,978	225.49	10	162,864	128.54
	0.5	3	922,581	469.00	8	363,339	241.47	9	278,917	193.42
1.0	0.05	10	49,061	36.43	10	1,429	1.47	10	880	0.89
	0.25	10	43,660	19.91	10	5,294	3.30	10	3,768	2.65
	0.5	8	461,000	194.60	9	127,764	73.63	9	102,612	70.01
1.5	0.05	10	1,300	1.55	10	153	0.21	10	118	0.18
	0.25	10	1,208	0.64	10	146	0.16	10	100	0.13
	0.5	9	138,632	60.01	10	34,249	19.99	10	3,765	2.39

Table 3.6 Comparison of efficiency of dominance rules based on Strategy III

α	β	(a)			(b)			(c)		
		#opt.	#nodes	Time	#opt.	#nodes	Time	#opt.	#nodes	Time
0.0	0.05	10	534	0.28	10	347	0.19	10	22	0.06
	0.25	0	1,890,758	600.01	6	820,988	435.77	10	13,124	9.63
	0.5	0	2,008,913	600.01	2	998,875	524.29	7	535,957	295.87
0.5	0.05	5	1,211,375	430.38	10	54,831	33.00	10	20,289	12.48
	0.25	3	1,864,982	516.12	8	450,469	232.48	10	225,682	115.90
	0.5	3	2,441,772	481.97	8	967,751	276.23	8	634,601	204.44
1.0	0.05	10	101,762	32.93	10	2,319	1.43	10	1,306	0.78
	0.25	10	218,247	27.06	10	11,446	2.98	10	8,159	2.42
	0.5	8	2,480,433	236.62	9	336,642	81.50	9	209,466	71.62
1.5	0.05	10	2,699	1.44	10	230	0.20	10	168	0.16
	0.25	10	1,827	0.46	10	169	0.14	10	116	0.10
	0.5	10	199,995	45.16	10	42,815	12.98	10	4,198	1.38

Tables 3.5 and 3.6 also show similar results with Table 3.4. Analysis of Variance (ANOVA) is conducted in order to analyze the effect of dominance rules and lower bounds effect on the number of nodes being searched by the branch-and-bound procedure and the results are presented in Table 3.7. Table 3.7 shows that *P*-Value (Lower bound) = 0.0001 and *P*-Value(Dominance rule) =0.0000 respectively, which are all smaller than $\alpha = 0.01$. Therefore we can conclude with strong evidence that lower bound strategy and dominance rules are significant factors that affect the branch-and-bound procedure performance.

Table 3.7 ANOVA for dominance rules and lower bounds

Source	Sum of Squares	D.F.	Mean Square	*F*-Ratio	*P*-value
Lower bound	3.24e+013	2	1.62e+013	9.9061	0.0001
Dominance rule	5.90e+013	2	2.95e+013	18.0350	0.0000
RESIDUALS	1.68e+014	103	1.64e+012		
TOTAL (CORRECTED)	2.59e+14	107			

3.5.3 Comparison of the Three Lower Bound Strategies

To compare the performance of the three lower bound strategies described in Section 3.4, computational experiments are also carried out based on randomly generated problem instances. The computational results for the branch-and-bound algorithm with both local and global dominance rules applied to problem instances with n = 10, 20 and 30 are given in Tables 3.8, 3.9 and 3.10 respectively. It is noted that a new set of problem instances with n = 20 has been randomly generated and thus these results would be independent from those in Tables 3.4, 3.5 and 3.6.

Table 3.8 Computational results for n = 10

α	β	Strategy I			Strategy II			Strategy III		
		#opt.	#nodes	Time	#opt.	#nodes	Time	#opt.	#nodes	Time
0.0	0.05	10	5	0.03	10	5	0.03	10	5	0.04
	0.25	10	48	0.01	10	45	0.02	10	46	0.13
	0.5	10	272	0.03	10	154	0.07	10	164	0.06
0.5	0.05	10	140	0.02	10	53	0.03	10	64	0.02
	0.25	10	198	0.03	10	49	0.04	10	59	0.03
	0.5	10	1,431	0.12	10	185	0.10	10	288	0.07
1.0	0.05	10	141	0.02	10	51	0.03	10	63	0.03
	0.25	10	218	0.02	10	56	0.03	10	63	0.02
	0.5	10	277	0.03	10	75	0.04	10	104	0.03
1.5	0.05	10	38	0.01	10	27	0.02	10	26	0.01
	0.25	10	29	0.01	10	19	0.01	10	21	0.02
	0.5	10	13	0.01	10	8	0.01	10	8	0.01

As shown in Table 3.8, all the problem instances with n = 10 can be solved almost instantaneously because of the small size of these instances. The average number of nodes shown in Table 3.8 indicates that Strategy I is likely to explore more nodes to find and verify optimal solutions than Strategies II and III. Similar results are also obtained in Tables 3.9 and 3.10. As the lower bounds obtained by Strategies II and III at the same level of the search tree are likely to be tighter than the lower bounds obtained by Strategy I based on Eq. (3.4) as described in Section 3.4, Strategy I may have to explore more nodes than Strategies II and III to find and verify the optimal solutions. Since Strategy III solves assignment problems only at the pre-specified levels of the search tree, the tightness of its lower bounds at the same search tree level lies between Strategy I and Strategy II, and hence it is reasonable that the average number of nodes explored also lies between that of Strategy I and Strategy II in general.

Table 3.9 Computational results for n = 20

α	β	Strategy I			Strategy II			Strategy III		
		#opt.	#nodes	Time	#opt.	#nodes	Time	#opt.	#nodes	Time
0.0	0.05	10	21	0.02	10	21	0.04	10	21	0.03
	0.25	10	13,738	3.26	10	11,652	11.26	10	12,055	9.48
	0.5	7	1,481,830	270.74	7	262,216	245.24	7	369,243	238.86
0.5	0.05	10	544,972	94.69	10	44,513	33.36	10	53,980	30.52
	0.25	6	2,230,043	345.37	9	159,085	134.28	9	230,491	127.16
	0.5	0	4,338,258	600.01	8	233,932	185.61	8	415,523	199.08
1.0	0.05	10	58,090	7.75	10	872	0.93	10	1,177	0.81
	0.25	10	80,634	12.43	10	3,845	3.29	10	6,753	2.57
	0.5	10	591,307	61.52	10	36,014	16.65	10	84,427	14.47
1.5	0.05	10	1,163	0.17	10	168	0.20	10	215	0.15
	0.25	10	620	0.15	10	114	0.26	10	226	0.22
	0.5	10	11	0.01	10	9	0.01	10	10	0.02

The computational results in Tables 3.9 and 3.10 show that the three lower bound strategies have different effects on the performance of the branch-and-bound algorithm for problem instances with different characteristics. For $\alpha = 0$ and

$\beta \in \{0.05, 0.25\}$, strategy I is the most efficient one among the three lower bound strategies when both the number of optimal solutions found and verified and the average computation time are considered. For $\alpha = 0$ and $\beta = 0.5$, strategies II and III are more efficient than Strategy I when considering the number of optimal solutions found and verified and the average computation time. For $\alpha \in \{0.5, 1, 1.5\}$, Strategy III in general is the most efficient one when compared with Strategies I and II.

Table 3.10 Computational results for n = 30

α	β	Strategy I			Strategy II			Strategy III		
		#opt.	#nodes	Time	#opt.	#nodes	Time	#opt.	#nodes	Time
0.0	0.05	10	159	0.11	10	159	0.56	10	159	0.52
	0.25	5	1,009,147	397.86	2	300,078	503.29	2	367,079	498.13
	0.5	0	2,082,874	600.01	0	486,087	600.01	0	738,627	600.01
0.5	0.05	0	2,318,068	600.01	2	415,905	516.11	2	605,595	515.15
	0.25	0	2,600,571	600.01	0	365,066	600.01	0	645,449	600.01
	0.5	0	3,114,820	600.01	0	339,162	600.01	0	880,342	600.01
1.0	0.05	7	1,411,320	241.90	8	69,430	130.82	8	207,907	133.83
	0.25	3	3,681,516	461.15	6	524,107	311.14	6	1,488,158	292.15
	0.5	5	2,606,561	302.88	5	831,444	302.77	6	1,786,525	242.63
1.5	0.05	10	14,646	1.92	10	606	1.35	10	5,692	4.97
	0.25	10	19,023	4.67	10	4,249	6.15	10	2,361	3.70
	0.5	10	1	0.03	10	1	0.03	10	1	0.03

The different efficiencies of the three strategies are due to the different characteristics of the problem instances. When α and β are small, all jobs have similar release dates and the variation of due dates is small. Therefore, all the jobs become both available and tardy rapidly after a few jobs are scheduled. As a result, stronger lower bounds are likely to be obtained at the upper levels of the search tree, and hence strategy I is efficient when both α and β are small. When the variation of due dates is large, it is possible that most of the early assigned jobs will not be tardy and the lower bounds obtained at the upper levels of the search tree by solving the

assignment problem are weak. Therefore, it is not likely that stronger lower bounds can be obtained at the upper levels and hence strategy II may be efficient. Strategy III on the other hand is a balance between Strategies I and II, and may be efficient for moderate variation of due dates. Thus, we conclude that different strategies should be applied to solve problems with different characteristics.

Table 3.11 Computational results of Akturk and Ozdemir (2000) for $n = 20$

α	β	n= 20		
		#opt.	#nodes[1]	Time (seconds)
0.0	0.05	10	352.7	0.7
	0.25	10	1977373.2	1569.3
	0.5	5	2902084.0	3071.3
0.5	0.05	10	712961.4	387.0
	0.25	10	1129594.0	1765.7
	0.5	7	1107718.1	1348.2
1.0	0.05	10	114840.5	160.9
	0.25	10	593646.4	1107.0
	0.5	10	443117.0	1336.7
1.5	0.05	10	66543.1	54.1
	0.25	10	6781.5	37.8
	0.5	10	61.1	0.1

1 – Average number of nodes. Maximum number of nodes = 4000000.

The branch-and-bound method of Akturk and Ozdemir (2000) was implemented using the GNU C compiler with the -02 optimizer option and ran on a SPARC Station 10 under SunOS 5.4. Their branch-and-bound algorithm will stop if optimal scheduled cannot be verified after the algorithm has reached the maximum node limit of 4000000. 10 random replications were generated based on the setting presented in Table 3.1 and the computational results for 20-job problems are presented in Table 3.11. For $\alpha = 0$ and $\beta = 0.05$, the Strategy III of the proposed branch-and-bound method can find and verify optimal solutions with an average running time of 0.03 seconds and average 21 nodes being searched. While the branch-and-bound

method of Akturk and Ozdemir (2000) had to enumerate an average of 352.7 nodes to find and verify all optimal solutions. For harder problems with $\alpha = 0.5$ and $\beta = 0.5$, the Strategy III of the branch-and-bound method proposed in this research work enumerated an average of 415523 nodes in an average of 199.08 seconds and proved 8 optimal solutions. Akturk and Ozdemir's (2000) method searched an average of 1107718.1 nodes in an average of 1348.2 seconds but only proved 7 optimal solutions. Since different hardware and platforms were used for computational experiments, it is difficult to compare the computation time without bias. However, the proposed branch-and-bound method generally enumerates less nodes than Akturk and Ozdemir's (2000) method to find and prove optimal solution

3.6 Conclusions

In this chapter, three global dominance rules and a simplified version of a local dominance rule for problem $1|r_i|\sum w_i T_i$ have been proposed. The computational experiments show the efficiency of the proposed dominance rules through a reduction in the size of the search tree and the computation time for solving the problem. An assignment problem-based lower bound computation method is also proposed and compared with another method that uses a general job splitting method to obtain the lower bound. Three lower bound implementation strategies are tested based on randomly generated problem instances and the computational results provide useful guidelines for the future use of the branch-and-bound algorithm to solve problem $1|r_i|\sum w_i T_i$. However, it is possible to improve the performance of the branch-and-bound algorithm by incorporating stronger and more efficient lower bound

computation methods and intelligent backtracking strategies, and these are areas of future research work.

Chapter 4 An Overlapped Neighborhood Search Algorithm for Sequencing Problems

In this chapter, an original heuristic, named overlapped neighborhood search (ONS) algorithm, is presented for single machine scheduling problems whose solutions can be represented with permutations. The ONS algorithm decomposes the sequence of a solution into small sized overlapped blocks; the solution space of each block is then explored independently. In addition, the ONS algorithm is a general-purpose algorithm that is able to solve a wide variety of sequencing problems, such as various single machine scheduling problems (SMSPs), the traveling salesman problem (TSP), linear ordering problems (LOP), quadratic assignment problems (QAP), and bandwidth reduction problems (BRP). To test the performance of the ONS algorithm, comprehensive computational experiments are carried out for SMSPs. Our computational results show that the ONS algorithm is efficient in solving these problems.

4.1 Introduction

Many decision problems encountered in manufacturing environments, such as various SMSPs, are formulated as combinatorial optimization problems and their solutions can be represented with permutations. Due to the computational complexity of combinatorial optimization problems - in particular, the large sized problems encountered in practice - the performance of exact algorithms are often poor as these problems are too difficult to be solved exactly within reasonable computation times. As a result, heuristics are developed to generate satisfactory solutions within reasonable computation time. If the solutions obtained using simple solution

construction heuristics are not satisfactory, one can often resort to local search heuristics to improve the existing solutions further. However, the main drawback of local search algorithms is that they are often trapped in local optimal solutions. This drawback has led to the consideration of algorithms that can guide the local search algorithms in getting out of traps and further improve the existing solutions.

To overcome the drawback of local search algorithms, researchers in the areas of operations research and artificial intelligence have introduced meta-heuristics that were applied successfully in solving many complex optimization problems. These meta-heuristics include a simulated annealing (SA) algorithm introduced by Kirkpatrick et al. (1983), tabu search (TS) algorithm proposed by Glover (1989, 1990), ant system (AS) algorithm developed by Colorni (1991), Dorigo et al. (1996), as well as the Greedy Randomized Adaptive Search Procedure (GRASP) implemented by Feo and Resende (1995), and Resende and Ribeiro (2003). However, the most efficient meta-heuristics often rely on the problem's information and can be viewed only as customized heuristics based on problem specific information, see Campos et al. (2005). Hence, these solution procedures cannot be separated from the optimization problem models.

The general purpose and problem independent algorithms are anticipated for combinatorial optimization problems. The advantage of general purpose and problem independent algorithms is that these algorithms can be applied to a wide variety of problems without modification of the fundamental models. The disadvantage of the problem independent algorithms is that they may be inferior to those of specialized procedures because the problem specific information is ignored.

We will present a general purpose algorithm for the single machine scheduling problem in this chapter. This chapter is organized as follows. Section 4.2 presents the details of the ONS algorithm. Several local search methods, which can be used to

explore the solution space of blocks, are provided in Section 4.3. The detailed implementation issues of the ONS algorithm for single machine scheduling problems with or without setup times are provided in Section 4.4. To illustrate the performance of the proposed algorithm, computational experiments based on SMSP instances were carried out and the computational results are presented in Section 4.5. Section 4.6 gives some concluding remarks.

4.2 Overlapped Neighborhood Search Algorithm

As a general purpose sequencing algorithm, the ONS algorithm explores the solution neighborhood and treats the objective function evaluation as a black box. The black box model for the ONS algorithm is illustrated in Figure 4.1. As the objective function is evaluated outside the ONS algorithm, the ONS algorithm does not know anything about how the solution is evaluated.

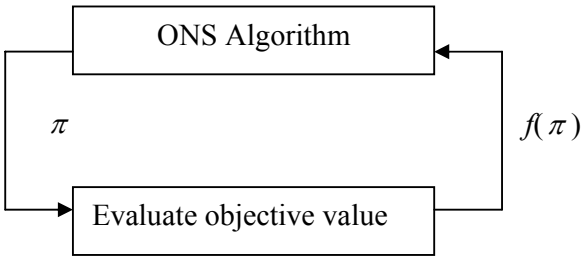


Figure 4.1 Black box model of ONS algorithm

As stated by Glover et al. (1993), the meta-heuristics often require a definition of the neighborhood. The use of large neighborhoods is attractive when searching for good solutions, but it is time consuming to explore the neighborhood fully. Smaller neighborhoods are both simpler and faster to explore, but they may not produce satisfactory solutions. Overall, it is preferable to use a small neighborhood if the quality of the solution obtained is reasonably good. For a sequencing problem, it is conjectured that it would be unlikely to improve an existing good solution very much

by relocating an object to positions far away from their original positions. The ONS algorithm proposed in this dissertation is based on this conjecture.

4.2.1 Overlapped Neighborhoods

The sequencing problem is to find a permutation $\pi = (p_1, p_2, \ldots, p_n)$ of the objects $\{1, 2, ..., n\}$ in order to minimize or maximize the objective function value, where p_i is the index of the object at position i. The general purpose ONS algorithm is a methodology that operates on a solution vector, which is one of the possible permutations of objects. The ONS algorithm divides an existing solution into blocks that overlap the blocks next to them; each block is then explored independently by utilizing block improvement procedures (BIPs) which are described in Section 4.3. The overlapped neighborhoods for a sequencing problem are illustrated in Figure 4.2. As shown in Figure 4.2, a sequence of the objects is divided into overlapped blocks B_1, B_2, B_3, and so on. There is an overlap between any two adjacent blocks.

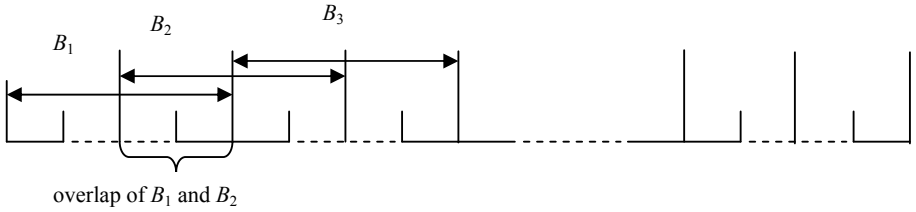


Figure 4.2 Illustration of the overlapped blocks

4.2.2 ONS Algorithm Framework

The existence of overlaps makes it possible that overlapped partial solutions are explored at least twice. This is equivalent to the intensification strategy of Tabu Search (TS), where the search focuses on the examination of elite solutions (Glover 1989). For the ONS algorithm, the intensification strategy is implemented by intensive search on the overlapped partial permutations. As the search on a block reaches a local

optimum and is improved by BIP, the previously searched block will be searched again for a possible improvement. Therefore, the overlap between any two blocks makes it possible to improve further the search on a particular block, as well as the whole solution, if the permutation of objects in its next block is changed. This procedure is repeated until no further improvement is possible. In this way, a large sized combinatorial problem can be improved based on its constituent small neighborhoods. For the example illustrated in Figure 4.2, if the local optimal solution of B_2 is obtained and the objective of the problem is improved, the ONS algorithm will backtrack to B_1 to implement BIP. If block B_2 cannot be further improved, BIP will proceed to explore block B_3. This backtrack search procedure is repeated until no improvement is possible. As the backtrack search procedure is a local search algorithm, it will terminate when the current solution cannot be further improved.

Let S_B denote the size of the block, and let S_O be the size of the overlap. The sizes of the block and overlap are the number of objects in the block and in the overlap respectively. For a sequencing problem of size n, a solution can be divided into $\lceil (n-S_B)/(S_B-S_O) \rceil + 1$ blocks. The ONS algorithm can also be implemented for all the possible S_B, where $S_B \in \{2,3,\ldots,n\}$. For $S_B = 2$, the ONS algorithm reduces to an adjacent pairwise interchange local search method (Baker 1974).

The computational requirements of the ONS algorithm can be reduced as follows. The BIP will backtrack to the recently searched block only when the permutation of objects in the overlapped neighborhood is changed; otherwise, it will move to the next block.

The search procedure described above is summarized as follows.

Step 1. Generate an initial solution;

Step 2. Apply BIP to the current block. If the solution is improved, go to Step 3; otherwise, go to Step 4;

Step 3. If the permutation of the block is changed and the current block is not the first block, set the previous block as the current block and go to Step 2; otherwise, go to Step 4;

Step 4. If the current block is the last block, stop; otherwise, move to the next block and go to Step 2.

4.3 Block Improvement Procedures

As the ONS algorithm is developed as a general purpose and problem independent algorithm, it is required that the BIPs are general procedures in order to explore the solution spaces of different problems. The criteria to design the block improvement strategies are:

(1) The block improvement method must be problem independent;

(2) The block improvement method should be able to explore the solution space as much as possible;

(3) The block improvement method should be computationally efficient.

We use two local search algorithms developed for the TSP to illustrate the difference between a problem independent algorithm and a problem dependent algorithm.

One of the most well known local search algorithms for the TSP is the 2-opt local search algorithm developed by Croes (1958), which is based on performing a move to improve a given solution. Each move consists of exchanging 2 edges from the current tour with 2 edges not in that tour as long as the result remains a tour. The 2-opt algorithm is a problem independent algorithm which does not utilize the distance

matrix information as it explores the solution neighborhood. A problem dependent algorithm restricts the exchange of edges by considering only those nodes that are relatively close to each other. For example, Steiglitz and Weiner (1968) proposed a restricted 3-opt method which stores, for each node i , a list of neighboring remaining nodes in an order of increasing distances from i. This truncated neighborhood is smaller than the original neighborhoods and the computation time can hence be reduced. However, the problem dependent information used in the restricted 3-opt local search method may not exist for other sequencing problems such as SMSPs. Hence, those problem specific algorithms cannot be embedded in the ONS algorithm. Several problem independent BIPs are provided in the subsection below.

4.3.1 Generalized Crossing (GC) Method

GC method was initially proposed by Zeng et al. (2007) for solving vehicle routing problems (VRP). The advantage of the GC method is that it is simple, fast, and it defines a large sized neighborhood. In this study, the GC method is adapted to solve the sequencing problem.

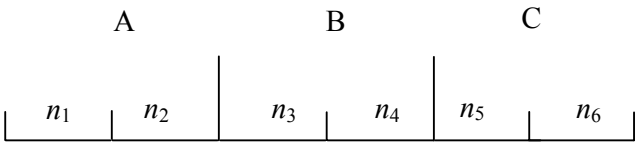


Figure 4.3 Initial sequence in a block

Unlike the VRP, the nodes in the sequencing problem are decomposed into three strings, represented by A, B, and C respectively, as illustrated in Figure 4.3. Five new partial sequences can be generated by reordering the three strings, as illustrated in Figure 4.4. As any of the three strings A, B, and C can also be reversed to form new sequences, 7 new sequences can be produced by reversing nodes in one string, two

strings, or three strings. Therefore, up to 48 different blocks (including the original block) may be generated from one original block.

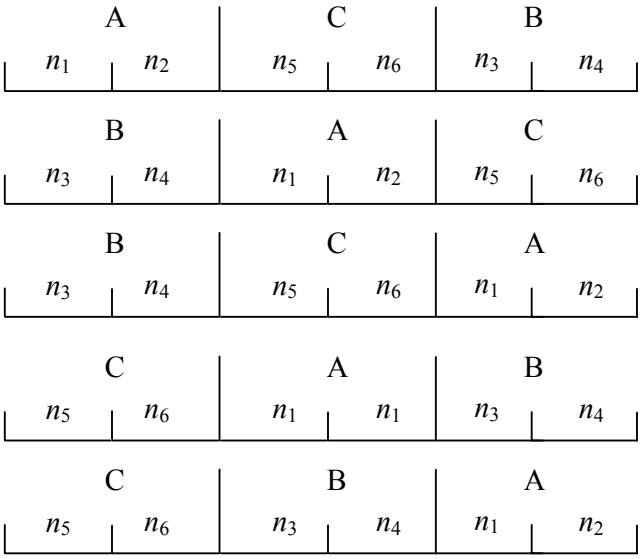


Figure 4.4 Sequences generated by re-sequencing three strings

4.3.2 Problem Independent Algorithms Developed for TSP

As shown in Figure 4.3, each block can be represented by a Hamiltonian path. In the mathematical field of graph theory, a Hamiltonian path is a path in an undirected graph that visits each vertex exactly once (Christofides 1970). It was shown by Christofides (1970) and Boffey (1973) that the problem of finding the minimal length Hamiltonian path is equivalent to finding the shortest tour of the TSP.

We will show how to convert the Hamiltonian path problem to a TSP problem by modifying the distance matrix. Here, we assume that the Hamiltonian path problem and the TSP are symmetric. For the asymmetric problem, the distance matrix will be modified accordingly. Let $N = \{1, 2, \ldots, n\}$ denote the set of objects in a block and let $C = [c_{ij}]$, where $i, j = 1, 2..., n$, denote the distance matrix respectively, and let M be a large positive number, e.g. greater than the sum of the matrix element values. For each

block defined in the ONS algorithm, let p and s denote the objects of a block immediately preceding and succeeding the current block (dummy objects may be required if the current block is the first block or the last block). We add vertices p and s as well as a dummy object d into N to form the vertex set $N' = \{p, s, d, 1, 2, \ldots, n\}$. The revised distance matrix C' is given below:

$$\begin{aligned}
&c'_{ij} = c_{ij} \quad \text{for } i, j \in \{1, 2, \ldots n\}, \\
&c'_{ip} = c'_{pi} = c_{ip} \quad \text{for } i \in \{1, 2, \ldots n\}, \\
&c'_{is} = c'_{si} = c_{is} \quad \text{for } i \in \{1, 2, \ldots n\}, \\
&c'_{di} = c'_{id} = 2M \quad \text{for } i \in \{1, 2, \ldots n\}, \\
&c'_{dp} = c'_{pd} = c'_{ds} = c'_{sd} = 0.
\end{aligned} \tag{4.1}$$

Under the transformation given in (4.1), the optimal or local optimal solutions to the TSP are guaranteed to contain the partial path $(p - d - s)$. Therefore, the solution to the TSP problem with vertex set $N' = \{p, s, d, 1, 2, \ldots, n\}$ and distance matrix C' is equivalent to the solution to the problem with vertex set $N = \{1, 2, \ldots, n\}$ and distance matrix C. As a result, any algorithm developed for the TSP can be applied to find the minimal length Hamiltonian path.

The above transformation of the Hamiltonian problem to the TSP is demonstrated based on the distance matrices C and C'. For those problems that do not have distance matrix C for the original problem, e.g. SMSP without setup times, C' can be defined as follows:

$$\begin{aligned}
&c'_{ij} = 0 \quad \text{for } i, j \in \{1, 2, \ldots n\}, \\
&c'_{ip} = c'_{pi} = 0 \quad \text{for } i \in \{1, 2, \ldots n\}, \\
&c'_{is} = c'_{si} = 0 \quad \text{for } i \in \{1, 2, \ldots n\}, \\
&c'_{di} = c'_{id} = 2M \quad \text{for } i \in \{1, 2, \ldots n\}, \\
&c'_{dp} = c'_{pd} = c'_{ds} = c'_{sd} = 0.
\end{aligned} \tag{4.2}$$

Based on (4.2), the optimal or local optimal solution to the TSP is also guaranteed to contain the partial path $(p-d-s)$.

One should note that the purpose of the problem transformation above is to explore the block's solution space instead of finding the minimal length of Hamiltonian path. The objective value of the block will be evaluated by the objective function defined for the original problem to be solved. Based on the previous discussion, it is obvious that any of the problem independent procedures developed for the TSP can be applied to explore the solution neighborhood. Some of these algorithms are 2-opt (Croes 1958), 3-opt, and r-opt (Lin and Kernighan 1973) local search algorithms as well as the Or-opt local search algorithms proposed by Or (1976).

4.3.3 Insertion and Interchange Based Local Search Procedures

The insertion and interchange based heuristics have been applied to many different optimization problems. In an insertion operation, an element at position *i* is inserted into another position *j* ($i \neq j$). Formally, the insertion operation for a permutation $\pi = (p_1, \ldots, p_{i-1}, p_i, p_{i+1}, \ldots, p_{j-1}, p_j, p_{j+1}, \ldots, p_n)$ is defined as:

$$\text{insert}(\pi, i, j) = \begin{cases} (p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_{j-1}, p_j, p_i, p_{j+1}, \ldots, p_n), & i < j \\ (p_1, \ldots, p_{j-1}, p_i, p_j, p_{j+1}, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n), & i > j \end{cases}.$$

The interchange local search procedure swaps the element at position *i* with the element at position *j,* for $i \neq j$, as shown below:

$$\text{Interchange}(\pi, i, j) = (p_1, \ldots, p_{i-1}, p_j, p_{i+1}, \ldots, p_{j-1}, p_i, p_{j+1}, \ldots, p_n).$$

It is noted here that we are not aiming to enumerate all the BIPs for the ONS algorithm. There are many other problem independent local search methods that can be embedded in the ONS algorithm, and the efficiency of the ONS algorithm may be affected by the BIP employed.

4.4. Implementation Issues

The ONS algorithm can be implemented in many variants. One of the typical implementation procedures is given here.

Step 1. Generate an initial solution, set $S_B := 3$, $S_O := \text{Coefficient} \times S_B$ ($0 < Coefficient < 1$), maximum size of block;

Step 2. Define the overlapped blocks based on the values of S_B, S_O and get the number of blocks that the initial solution is divided into; set Current Block Index: = 1;

Step 3. If Current Block Index = 0, reset Current Block := 1; if Current Block is greater than the number of blocks obtained in Step 2, go to Step 8;

Step 4. Implement BIP within the current block until no improvement can be made;

Step 5. If the current block is improved in Step 4, go to Step 6; otherwise, go to Step 7;

Step 6. Set Current Block Index := Current Block Index - 1, go to Step 3;

Step 7. Set Current Block Index := Current Block Index +1, go to Step 3;

Step 8. Increase S_B by a step size. If S_B is less than the maximum size of block, set $S_O := Coefficient \times S_B$ and go to Step 2; otherwise, go to Step 9.

Step 9. Stop.

The general procedure described above is able to explore different sizes of blocks iteratively. There are several control parameters in the ONS algorithm. They are the size of block S_B, the size of overlap S_O, and the step size of the increment of S_B in Step 8. In general, the ONS algorithm will produce a better solution with a large S_O and a small step size of the increment of S_B, albeit with the cost being longer computation times.

Based on the previous discussion in this chapter, we can see that the ONS algorithm developed in this research work is different from existing local search algorithms and meta-heuristics. Firstly, it is a general purpose heuristic for a wide

variety of sequencing problems. Any problem whose solution can be represented by permutations can be solved by the ONS algorithm. Secondly, many existing local search methods can be used as BIPs for the ONS and hence makes ONS an expandable heuristic. Thirdly, the ONS algorithm is able to explore problem solution space of each block independently and enlarge the solution space, which makes it possible to find a good solution. Moreover, the computation complexity of the ONS algorithm is controllable by setting different sizes of blocks and overlaps. This feature makes it suitable to solve different size of problems with controllable computation time.

4.5 Computational Experiments

To test the performance of the proposed ONS algorithm, computational experiments were carried out based on TSP benchmark problem instances and SMSP instances. The ONS algorithm was coded in C++ and all computational experiments were conducted on a Pentium 4 PC with 2.6 GHz CPU with 512MB RAM running the Windows XP operating system. The computation time reported is in seconds.

4.5.1 Computational Experiments for the SMSP with Unequal Release Dates

In a single machine scheduling problem with the objective of minimizing the total weighted tardiness, n independent jobs are released continuously and each job has a processing time p_i, a release date r_i, a due date d_i, and a tardiness penalty weight w_i. Compared to the TSP, the evaluation of the SMSP neighbors is more time consuming as the weighted tardiness of all the jobs following the first job that is moved forward have to be updated. The GC method is applied as the BIP for the SMSP with unequal release dates. Our computational experiments for the SMSP were conducted based on

the random problem instances generated using the scheme in Akturk and Ozdemir (2001).

Each instance is generated from four uniformly distributed parameters of r_i, p_i, d_i, and w_i. The values of p_i and w_i are all uniformly distributed between a lower bound value and an upper bound value. The distributions of r_i and d_i depend on two parameters: α and β. For each job, r_i is generated from the uniform distribution $\left[0, \alpha\sum p_i\right]$ and $d_i - (r_i + p_i)$ is generated from the uniform distribution $[0, \beta\sum p_i]$, where $\alpha \in \{0, 0.5, 1, 1.5\}$ and $\beta \in \{0.05, 0.25, 0.5\}$. The settings of the problem generating parameters are shown in Table 4.1. For each combination of settings, 20 random instances were generated and hence 2880 random problem instances were generated in total.

The relative improvement in percentage is defined in Akturk and Ozdemir (2001) as:

$$\text{Impr.} = \begin{cases} \left(WT^h - WT^{\text{Improve}}\right) / WT^h \times 100 & \text{if } WT^h > 0 \\ 0 \quad \text{otherwise} \end{cases},$$

where WT^h is the total weighted tardiness value of the initial schedule obtained by heuristic rules and WT^{Improve} is the total weighted tardiness value obtained using an improvement algorithm.

Table 4.1 Problem generating parameters

Factors	Number of levels	Setting
Number of jobs	3	50, 100, 150
Variability of p_i	2	[1, 10], [1, 100]
Variability of w_i	2	[1, 10], [1, 100]

The ATC heuristic rule developed by Rachamadugu and Morton (1981), the weighted short processing time rule (WSPT), and the weighted earliest due date (WDD)

rule are used to generate initial solutions. Of the three heuristic rules, it was shown by Akturk and Ozdemir (2001) that the WDD and WSPT rules performed poorly compared to the ATC since these two rules does not consider the unequal release dates. As the three heuristic rules showed different efficiency in producing initial schedules, we were able to analyze the sensitivity of the ONS algorithm to the quality of the initial schedules. For the ATC rule, a fixed value of $k = 1$ was used throughout the computational experiments because our preliminary computational experiments showed that $k = 1$ generated slightly better results compared to other values.

Compared with the local dominance rule (LDR) proposed by Akturk and Ozdemir (2001), which produces schedules that cannot be improved by adjacent pairwise interchange (API), the ONS algorithm is able to produce larger sizes of neighborhoods due to the larger size of neighborhoods generated by the GC BIP. Even when $S_B = 3$, $S_O = 1$, the neighbors generated by the LDR local search method is only a subset of the neighbors generated by the GC BIP.

Table 4.2 Computational results of ONS and LDR

		LDR			ONS		
n	Method	Average tardiness[1]	Average Impr.%[2]	Time[3]	Average tardiness[1]	Average Impr.%[2]	Time[3]
50	ATC	114118	12.0	0.026	113708	13.5	0.027
	WSPT	133957	28.7	0.057	130916	32.9	0.059
	WDD	120220	41.0	0.125	117586	44.4	0.123
100	ATC	416150	13.7	0.050	415274	15.3	0.053
	WSPT	489644	28.8	0.081	480071	33.7	0.167
	WDD	446481	43.7	0.666	435284	48.6	0.635
150	ATC	958655	16.8	0.126	957136	18.5	0.109
	WSPT	1118483	27.6	0.181	1102218	32.6	0.149
	WDD	1037425	44.5	1.134	1007544	49.2	1.396

1 – The average tardiness of 960 problem instances
2 – The average improvement in percentage over 960 instances
3 – The average computation time for each replication based on PC with 2.6GHz CPU and 512MB RAM in seconds

The computational results of the ONS algorithm with $S_B = 3$, $S_O = 1$ and the LDR local search method developed by Akturk and Ozdemir (2001) for 50, 100 and 150 jobs are summarized in Table 4.2. For each of the three heuristic rules, the average tardiness, the average improvement in percentage, and the average computation time over the 960 runs were reported. The computation time reported in Table 4.2 is the time taken for performing the local search only; furthermore, it is only a rough estimation of the time taken, since the computing time is too small to be measured accurately.

The corresponding results in columns 4 and 7 in Table 4.2 show that the ONS method tends to outperform LDR for different sizes of problems based on different initial solutions. The computation time taken by the ONS algorithm is similar to the computation time taken by the LDR method.

For the success of the ONS algorithm, it is conjectured that it is unlikely to improve a good existing solution by relocating an object to positions far away from its original position. In order to validate this conjecture, the performance of the ONS algorithm was further evaluated with different sizes of blocks and overlaps for problem instances with $n = 100$. The sizes of blocks were set as $S_B \in \{3, 8, 32, 64, 96\}$ with various sizes of overlap. It has to be pointed out that when the size of the block is small, the relocation of jobs is limited to those positions that are within the vicinity of their original positions. However, when the size of the block is large, it is possible to relocate a job to positions that are both within the vicinity of their original positions, and also to positions that are far away. As it is shown that the ATC rule can produce good initial schedules and good improved schedules, the following computational experiments will use the ATC heuristic rule to generate the initial schedules. The

detailed computational results with different parameter settings are summarized in Table 4.3.

Table 4.3 The average improvement in percentage for $n = 100$

α	β	$S_B = 3$		$S_B = 8$			$S_B = 32$			$S_B = 64$			$S_B = 96$		
		$S_O = 1$	$S_O = 2$	$S_O = 2$	$S_O = 4$	$S_O = 6$	$S_O = 8$	$S_O = 16$	$S_O = 24$	$S_O = 16$	$S_O = 32$	$S_O = 48$	$S_O = 24$	$S_O = 48$	$S_O = 72$
0.0	0.05	0.03	0.04[1]	0.03	0.03	0.04[1]	0.01	0.02	0.03	0.01	0.01	0.02	0.01	0.01	0.01
	0.25	0.10	0.10	0.06	0.08	0.11[1]	0.02	0.04	0.07	0.05	0.07	0.08	0.01	0.01	0.01
	0.50	0.22	0.25	0.17	0.34	0.43	0.74	0.96	1.35	0.94	1.09	1.39[1]	0.80	0.93	0.90
0.5	0.05	2.40	2.48	2.36	2.64	3.35[1]	0.98	1.04	1.60	0.56	0.85	1.14	0.51	0.62	0.98
	0.25	1.36	1.72	2.15	2.33	3.06	3.00	3.71	5.21[1]	2.14	3.32	4.61	2.19	2.38	3.51
	0.50	1.93	2.08	2.05	2.77	3.45	6.11	7.50	9.15	6.02	11.40[1]	11.02	7.17	7.95	9.70
1.0	0.05	14.51	16.10	11.09	15.45	20.42[1]	3.70	6.17	9.18	1.74	3.18	3.84	1.42	1.93	1.99
	0.25	42.48	46.43	42.41	44.67	49.68[1]	26.58	34.29	39.54	10.79	18.61	27.39	6.98	10.58	14.15
	0.50	36.56	40.28	40.72[1]	40.72[1]	40.72[1]	28.27	33.53	35.32	14.46	23.86	27.86	11.70	12.11	13.24
1.5	0.05	32.01	35.79[1]	19.57	25.55	35.28	6.05	8.07	17.40	3.13	4.72	5.22	2.96	2.90	2.82
	0.25	33.24[1]	33.24[1]	25.90	32.78	32.72	11.71	22.00	23.91	4.53	8.00	14.39	5.98	4.71	4.28
	0.50	19.29[1]	19.29[1]	18.04	16.79	18.04	8.54	10.31	13.25	9.04	9.59	9.23	4.76	7.26	4.76

1 – The largest average improvement obtained by the ONS method with different parameters

The computational results in Table 4.3 show that with an increase of the sizes of overlap, there is a general increase in average improvement. This is logical as larger overlap sizes will increase the solution space, and it is therefore possible to find better solutions. The computational results in Table 4.3 also show that the largest improvement is likely to be obtained with smaller sizes of blocks and larger sizes of overlaps.

As the sizes of the blocks are not evenly distributed as seen from Table 4.3, more computational experiments were carried out to determine how to choose the appropriate size of blocks for the ONS algorithm. The additional sizes of blocks tested were $S_B \in \{5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$ with the sizes of overlaps being $S_O = \lceil 3 \times S_B / 4 \rceil$. Here we set $S_O = \lceil 3 \times S_B / 4 \rceil$ to try to balance the computation time with the quality of the final schedules.

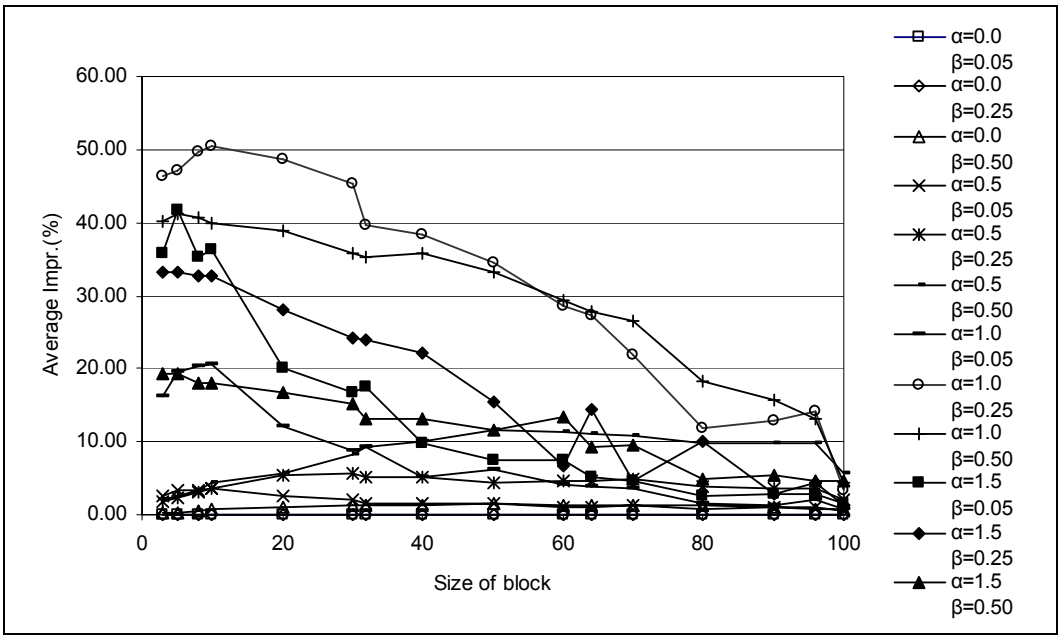


Figure 4.5 Average improvements for problems with different characteristics

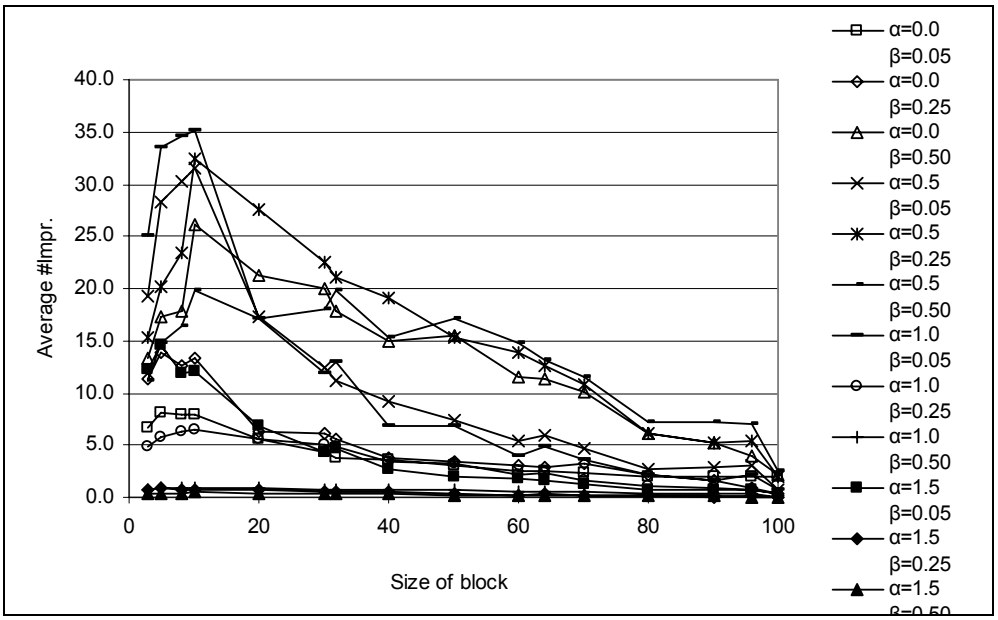


Figure 4.6 Average number of improvements for problems with different characteristics

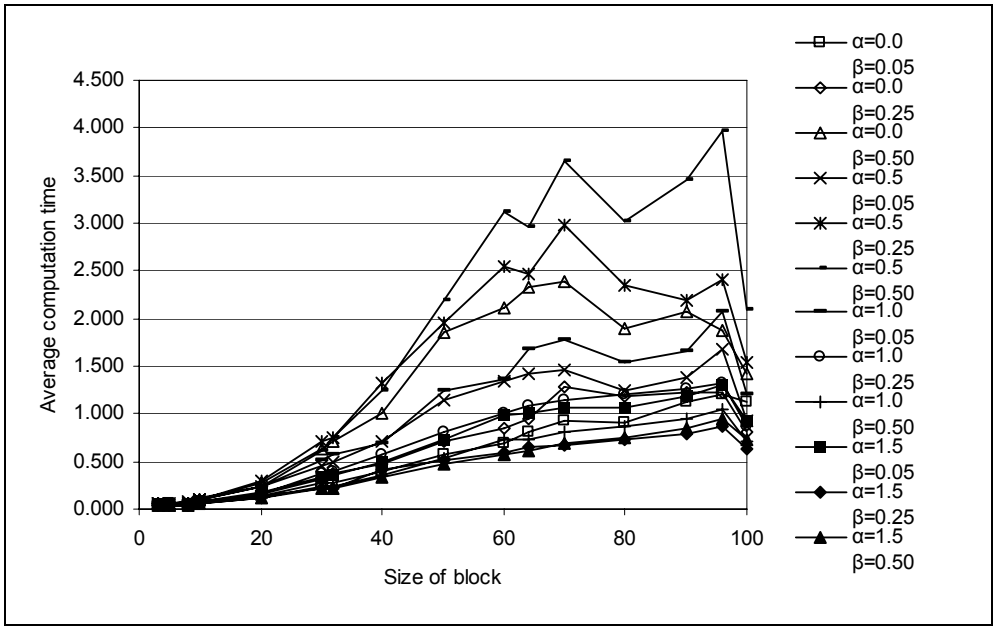


Figure 4.7 Average computation time for problems with different characteristics

The average improvement in percentage, average number of improvements, and average computation time based on different sizes of blocks for problems with different characteristics are presented in Figures 4.5, 4.6 and 4.7 respectively. Figure

4.5 shows that the average improvement is small with $\alpha = 0$ and is larger with $\alpha = 1, 1.5$. The maximum average improvement is obtained for most of the problems when $S_B = 10$ except for $\alpha = 0, \beta = 0.5$ and $\alpha = 0.5$, $\beta \in \{0.25, 0.5\}$. When the size of the block is very large, i.e. $S_B = 100$, the local search method becomes very inefficient. Figure 4.5 also shows that the efficiency of the ONS method is not sensitive to the size of the blocks when $S_B \leq 10$ as the difference of the average improvement is small when the size of the blocks changes. From Figure 4.6, we can see that $S_B = 10$ also gives the maximum average number of improvements for most of the problems and that the average number of improvements will decrease significantly when the size of the block increases. Comparing Figures 4.5 and 4.6, we can see that the average improvements and the average number of improvements follow the same trends as the size of the block changes. It is noted that the maximum average number of improvements in Figure 4.6 is obtained for $\alpha = 1.0, \beta = 0.05$ while the maximum average improvement is obtained for $\alpha = 1.0, \beta = 0.25$. This discrepancy comes about because the average improvement not only depends on the absolute reduction of total weighted tardiness, but also on the total weighted tardiness of the initial schedule, which may be substantially different for problems with different characteristics.

The results of the average computation times are presented in Figure 4.7. With the increase of the size of the block, the average computation time increases and subsequently decreases. As the GC method can generate $S_B(S_B - 1)/2$ strings for each block and the size of overlap is $S_O = \lceil 3 \times S_B / 4 \rceil$ in our computational experiments, the total number of strings that is explored by the GC method is approximately equal to $\left[\lceil (n - S_B)/(S_B - S_O) \rceil + 1\right] \times S_B(S_B - 1)/2 \approx (4nS_B - 4S_B^2 + 1)(S_B^2 - S_B)/2.$ The

relationship between the total number of strings being explored and the size of the blocks for $n = 100$ is illustrated in Figure 4.8. Figure 4.8 provides the relationship between the number of strings being explored and the size of the blocks. It can be seen that the computation time is proportional to the number of strings being explored. Moreover, it can be seen in Figure 4.7 that the computation time for problems with $\alpha = 0, \beta = 0.5$ and $\alpha = 0.5, \beta \in \{0.25, 0.5\}$ increases faster than the computation times for those problems with other characteristics. One possible reason is that for $\alpha = 0, \beta = 0.5$ and $\alpha = 0.5, \beta \in \{0.25, 0.5\}$, there are more average number of improvements that incur additional backtrack, hence increasing computation times rapidly. For some problems, i.e. $\alpha = 0.5, \beta = 0.5$, the computation time increases faster compared to the computation times for other problems due to the backtrack incurred.

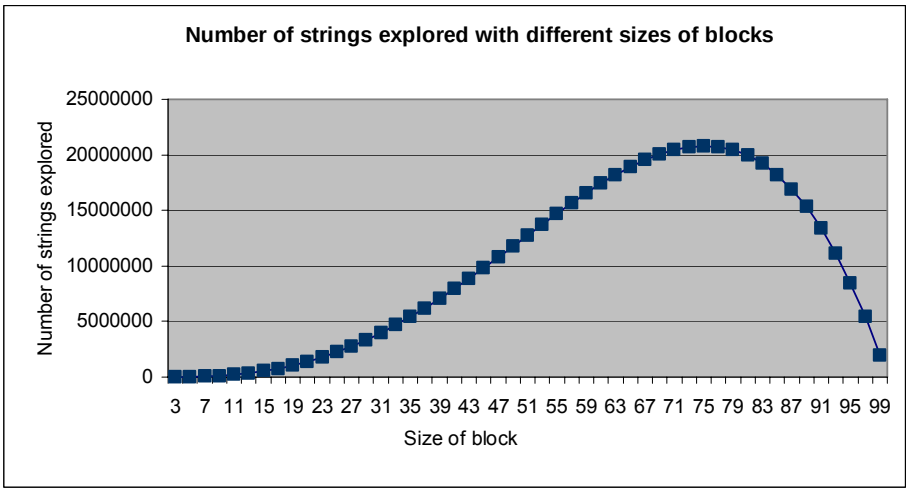


Figure 4.8 Number of strings explored with different sizes of blocks

Figures 4.5 and 4.6 show that it is not likely to improve an existing good solution significantly by relocating jobs to positions far away from their original positions. Based on the previous analysis, we were able choose the appropriate parameters for the ONS algorithm for problems with different characteristics. However, it is difficult to

choose the appropriate parameters when the problem characteristics are unknown. In this situation, the ONS algorithm can be run iteratively with different sizes of blocks. As it is likely to improve a schedule with small sizes of blocks within short a running time, as shown in Figures 4.5 and 4.7, it is preferable to implement the ONS algorithm with small sizes of blocks. Our implementation of the ONS algorithm is set as follows: the size of block is set to r times of the previous size of block until the size of block exceeds the size of the problem. For example, with a starting size of a block being 3 and $r = 2$, $n = 100$, the set of sizes of blocks is $S_B \in \{3, 6, 12, 24, 48, 96\}$ with the sizes of overlaps being $S_O = \lceil 3 \times S_B / 4 \rceil$ except for $S_B = 3$ with $S_O = 2$.

The computational results of the iterative ONS are presented in Table 4.4. The average improvement seen in Tables 4.3 and 4.4 shows that the iterative ONS algorithm always outperforms all the ONS methods with single size of block and the computation time is also acceptable. The computational results for the SMSP with unequal release dates indicate that the improvement of the quality of the solution is significant when using the iterative ONS algorithm, albeit at the expense of longer computation times.

Table 4.4 Computational results for iterative ONS

α	β	Impr. (%)[1]	#Impr.[2]	Time[3]
0.0	0.05	0.05	9.19	1.479
	0.25	0.16	14.89	1.632
	0.50	1.64	39.01	2.281
0.5	0.05	4.34	37.01	1.653
	0.25	7.79	48.55	2.414
	0.50	13.99	36.25	2.765
1.0	0.05	26.00	43.05	1.890
	0.25	55.85	8.04	1.429
	0.50	41.08	0.71	1.342
1.5	0.05	45.22	15.73	1.514
	0.25	34.78	0.80	1.331
	0.50	19.29	0.33	1.325

1- Average improvement from initial solution in percentage
2- Average number of improvement achieved
3- Average computation time in seconds

4.5.2 Computational Experiments for the SMSP with Sequence Dependent Setup Times

In a single machine scheduling problem with sequence dependent setup times, n jobs have to be sequenced on a machine to minimize the total tardiness. Let p_i, d_i denote the processing time and the due date of job i respectively, and let s_{ki} denote the setup time when job i succeeds job k immediately, where $i = 1, \ldots, n$. The tardiness of job i is denoted by T_i and T_i is defined as $T_i = \max\{C_i - d_i, 0\}$, where C_i is the completion time of job i. For this problem, it is assumed that all the processing times, due dates, and setup times are non-negative integers. In addition, job preemptions are not allowed. According to the standard scheme introduced by Graham et al. (1979), the SMSP with sequence dependent setup times minimizing the total tardiness of jobs is represented as $1|s_{ki}|\sum T_i$. Problem $1|s_{ki}|\sum T_i$ is NP-hard as even its relaxed problem, problem $1||\sum T_i$, is shown to be NP-hard by Du and Leung (1990). Since this problem is NP-hard, it is unlikely for any algorithm to always find an optimal solution within polynomial computation times.

In this subsection, the ONS algorithm is hybridized with GRASP to test its performance based on the SMSP with sequence dependent setup times. GRASP is a meta-heuristic which has been applied successfully to solve a variety of combinatorial optimization problems. It is a multi-start method having two phases, a construction phase and an improvement phase. The two phases are repeated a number of times and the best solution found is reported as the final solution. The detailed implementation of GRASP is described in Feo and Resende (1995), Resende and Ribeiro (2003), and Fernandes and Ribeiro (2005).

Fernandes and Ribeiro (2005) pointed out that path relinking is important to the basic GRASP and described two strategies to implement path relinking. The first strategy is to apply path relinking to a GRASP local optimal solution with a randomly selected elite solution; the second strategy is post-optimization, in which the elite solutions are connected via path relinking. Path relinking generates new solutions by exploring the trajectories that connect elite solutions. Starting from one of the elite solutions, called the initiating solution, a move is made in the neighborhood space of the initiating solution toward another solution, called the guiding solution. This is accomplished by selecting moves that introduce the attributes that are contained in the guiding solution but not in the initiating solution. These moves can be any of the neighborhood search moves, such as pairwise interchange, forward insertion, or backward insertion. As pointed out by Gupta and Smith (2006), the feasibility of the trial solutions after some or all attributes of the guiding solution have been included must be preserved and the attributes in the guiding solution must be introduced into the initiating solution. This process continues until all the attributes in the guiding solution are inherited by the initiating solution, that is, the initiating solution is the same as the guiding solution. Based on the procedure described above, it is possible that better solutions are found during the moves from the initiating solution to the guiding solution. In this study, the path relinking is used as a post-optimization mechanism. Path relinking procedure is applied to every pair of elite solutions as a post-optimization method to improve the elite solutions further. If a solution that is better than the current incumbent solution is found, the current incumbent solution is replaced by this solution. For path relinking, moves are made based on pairwise interchange, and forward or backward insertion.

The steps of the GRASP algorithm with path relinking are as follows.

Step 1. Initialization. Set Iteration := 0, MaxIter;

Step 2. Construct a solution;

Step 3. Improve the solution obtained in Step 2 by ONS algorithm until local optimal solution is reached. Set Iteration := Iteration + 1;

Step 4. If the local optimal solution obtained in Step 3 is better than the incumbent solution, replace the incumbent solution by the local optimal solution;

Step 5. If Iteration $\leq$ MaxIter, go to Step 2; otherwise, go to Step 6;

Step 6. Perform path relinking post-optimization and report the current incumbent solution. Stop.

The ATCS rule proposed by Lee et al. (1997) is applied to compute the job scheduling priority index. At the initial stage, the set of candidate jobs consists of all the jobs to be sequenced. The Restricted Candidate List (RCL) is constructed based on the greedy function, which is the ATCS priority index function in this study. Let I_{max} and I_{min} denote the maximum and minimum priority indices at time t over all the candidate jobs. The number of jobs in the RCL is determined by a threshold parameter $\alpha \in [0, 1]$. All the jobs in the candidate jobs with priority indices greater or equal to $R_{\max} - \alpha(R_{\max} - R_{\min})$ are included in the RCL. Thus $\alpha = 0$ corresponds to the pure greedy function while $\alpha = 1$ corresponds to a random job selection. A feasible solution is built in the construction phase by randomly selecting the next job according to a uniform distribution from the RCL at each reiteration, until all the jobs are included in the solution.

It is claimed by Gupta and Smith (2006) that the threshold parameter α is very important in the construction phase. As pointed out in Resende and Ribeiro (2003), it

is preferable to vary the value of α dynamically in order to obtain better final results. The value of α is chosen randomly from a uniform distribution [0, 1] in this study.

To test the performance of the ONS algorithm hybridized with the GRASP, denoted as GRASP+ONS, comprehensive computational experiments were conducted. In this study, a maximum number of 40 elite solutions were retained and the GRASP+ONS algorithm was run 20 times with different random seeds for each problem instance. The computational experiments were carried out based on test problem instances widely used in the literature. Some or all of the testing problem instances used in this computational experiment have also been used by Ragatz (1993), Rubin and Ragatz (1995), Tan et al. (2000), Gupta and Smith (2006), and Armentano and de Araujo (2006).

The testing problem instances consist of two sets. The first set consists of problem instances with 15, 25, 35 and 45 jobs, and is referred to as a small problem set. The second set consists of problem instances with 55, 65, 75 and 85 jobs, and is referred to as a large problem set. The problem instances of each size are derived from a $2 \times 2 \times 2$ experimental design related to three problem factors which are set at two levels. These three factors are the processing time variance (*PTV*) of the jobs, the tardiness factor (*TF*), and the due date range (*DR*). Each of these three factors is set at two levels. L and H stand for low and high levels for *PTV* and *TF*, respectively, while *N* and *W* stands for narrow and wide due dates respectively, as shown in Table 4.5.

The computational results based on the testing problem instances are presented in Tables 4.6 and 4.7 for both the small problem set and the large problem set respectively. The column BnB (denoting "branch and bound") in Table 4.6 is the objective value obtained by the branch-and-bound method reported by Ragatz (1993). The branch-and-bound algorithm is limited to the exploration of two million nodes and

optimal solutions were obtained for only some of the problems. For the small problem set, the results are represented in relative deviation from the BnB. The relative deviation in percentage is calculated using the function below,

$x = \left(\left(\text{alg.} / \text{BnB} \right) - 1 \right) \times 100\%$,

where alg. denotes the objective value of the heuristic. If the objective value obtained by BnB is equal to zero, the relative deviation is set to zero.

Table 4.5 Experimental design of problem instances

Problem no.	*PTV*	*TF*	*DR*
1	*L*	*L*	*N*
2	*L*	*L*	*W*
3	*L*	*H*	*N*
4	*L*	*H*	*W*
5	*H*	*L*	*N*
6	*H*	*L*	*W*
7	*H*	*H*	*N*
8	*H*	*H*	*W*

The computational results of the GRASP+ONS algorithm were compared with the results of the ACO algorithm proposed by Gagné et al. (2002) and the GRASP algorithm developed by Gupta and Smith (2006). For the small problem set, the best, median, worst solutions, and computation times are presented. Besides the best, median, worst solutions, and the computation time, the average results are also reported for the large problem set.

For the small problem set, 31 best solutions obtained by ACO were better or equal to the branch-and-bound, denoted as BnB, solutions among the 32 test problem instances. The best solutions obtained by GRASP and GRASP+ONS were all better or equal to the BnB solutions. For the 15-job problem instances, ACO, GRASP and GRASP+ONS had the same performance in terms of the best solutions found. For the seventh 25-job problem instances, the best solution obtained by ACO has a 7% deviation from the BnB solution, while GRASP and GRASP+ONS found the same

solution as that obtained by the BnB. For the 35-job problem instances, ACO, GRASP and GRASP+ONS obtained better solutions than the branch-and-bound solutions. Moreover, GRASP found 4 better solutions than the ACO, and GRASP+ONS found 3 better solutions than the ACO in terms of the best solution found. For the 45-job problem instances, GRASP+ONS obtained 5 better solutions than the ACO while GRASP obtained 4 better solutions than the ACO in terms of the best solutions found. The median and worst solutions among 20 runs for each problem instance are also presented in Table 4.6. In general, GRASP+ONS and GRASP outperformed the ACO in terms of the median and worst solutions. Table 4.6 also provides the computation times taken by the ACO, GRASP and GRASP+ONS. However, the computation times can only be compared approximately as different hardware platforms were used to conduct the computational experiments.

Table 4.7 presents the computational results of the ACO, GRASP and GRASP+ONS for the large problem set. Both GRASP+ONS and GRASP have 13 better solutions than the ACO in terms of the best solutions. For the problem instances with low *PT*, low TF, and narrow due date range, both the ACO and GRASP+ONS outperformed the GRASP. In terms of median and worst solutions, GRASP+ONS and GRASP outperformed the ACO in general. It is noted that the best solution reported in Gagné et al. (2002) for the first 75-job problem is 63. However, this value seems incorrect compared with the lower bound obtained by the branch-and-bound technique of Ragatz (1993).

Table 4.6 Comparison of experimental results for small problem set

Problem No.			ACO				GRASP				GRASP+ONS			
	# Jobs	BnB	Best	Median	Worst	Time[1]	Best	Median	Worst	Time[2]	Best	Median	Worst	Time[3]
1	15	90*	0.0	4.4	7.8	1.3	0.0	**0.0**	**0.0**	4.0	0.0	3.3	4.4	3.3
2	15	0*	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	15	3418*	0.0	1.1	2.1	1.5	0.0	**0.0**	**0.0**	3.7	0.0	**0.0**	**0.0**	3.9
4	15	1067*	0.0	0.0	0.0	1.4	0.0	0.0	0.0	2.5	0.0	0.0	0.0	3.5
5	15	0*	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	15	0*	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	15	1861*	0.0	0.0	1.1	1.5	0.0	0.0	**0.0**	3.9	0.0	0.0	**0.0**	3.0
8	15	5660*	0.0	1.1	1.5	1.5	0.0	**0.0**	**0.0**	3.2	0.0	**0.0**	**0.0**	3.1
1	25	264	-1.1	0.8	1.9	7.2	-1.1	-0.4	**-0.4**	14	-1.1	**-0.8**	**-0.4**	19.5
2	25	0	0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	25	3511	-0.4	0.3	0.9	7.8	-0.4	**-0.4**	**-0.4**	18.5	-0.4	**-0.4**	**-0.4**	21.5
4	25	0*	0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1
5	25	0*	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.4
6	25	0*	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	25	7225	0.7	1.8	3.7	9.8	**0.0**	**0.0**	**0.0**	24.4	**0.0**	**0.0**	0.4	21.1
8	25	2067	-5.9	5.9	14.2	8.6	**-7.4**	**-7.4**	**-7.4**	23.3	**-7.4**	**-7.4**	**-7.4**	18.6
1	35	30	**-46.7**	-13.3	6.7	29.8	**-46.7**	**-20.0**	**-3.3**	53.3	-33.3	3.3	33.3	75.3
2	35	0*	0.0	0.0	0.0	0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1
3	35	17774	-0.5	0.1	0.3	32.2	**-1.1**	**-0.9**	**-0.8**	94.4	-0.9	-0.5	-0.1	88.4
4	35	19277	-0.8	0.3	1.3	32.2	**-0.9**	**-0.8**	**-0.6**	88.8	-0.7	-0.5	-0.1	71.2
5	35	291	-15.1	-8.8	-1.0	31.0	**-16.5**	**-14.1**	**-13.4**	59.0	**-16.5**	-11.3	-4.1	76.0
6	35	0*	0.0	0.0	0.0	0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1
7	35	13274	-1.4	-0.1	0.6	27.9	**-2.3**	**-2.2**	**-1.8**	88.0	**-2.3**	-2.0	-1.2	71.2
8	35	6704	-29.4	-24.8	-20.1	33.0	-29.4	**-29.4**	**-29.4**	83.5	-29.4	**-29.4**	-29.3	63.9
1	45	116	**-11.2**	**-5.6**	**0.0**	83.2	**-11.2**	0.4	2.6	122.5	-1.7	3.4	9.5	183.6
2	45	0*	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1
3	45	27097	-1.6	-1.0	-0.5	91.8	-1.8	**-1.6**	**-1.3**	216.4	**-2.1**	-1.5	-1.1	226.5
4	45	15941	-2.8	-1.1	-0.3	89.2	**-4.6**	**-4.6**	**-4.4**	201.3	**-4.6**	-4.0	-3.7	170.9
5	45	234	-5.1	5.6	15.4	77.6	-5.1	**5.1**	**6.8**	129.9	**-7.7**	7.3	10.3	191.4
6	45	0*	0.0	0.0	0.0	0.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1
7	45	25070	-4.2	-3.3	-2.9	78.6	-5.0	**-4.6**	**-4.4**	253.3	**-5.1**	**-4.6**	-4.1	187.6
8	45	24123	-3.2	-1.7	-0.6	84.7	**-5.5**	**-5.4**	**-5.3**	267.0	**-5.5**	-4.9	-4.3	165.7

* - Optimal solution objective values obtained by the branch-and-bound algorithm proposed by Ragatz (1993)

1- Intel Pentium III 733MHz CPU with 256MB RAM personal computer

2- Intel Pentium IV 2.4GHz CPU with 1 GB RAM personal computer

3- Intel Pentium IV 2.6GHz CPU with 512MB RAM personal computer

Table 4.7 Comparison of experimental results for large problem set

Problem No.		ACO					GRASP					GRASP+ONS				
	Jobs	Best	Median	Average	Worst	Time	Best	Median	Average	Worst	Time	Best	Median	Average	Worst	Time
1	55	**212**	**237.5**	**241.3**	273	167.6	242	263.0	260.4	**269**	258.2	242	275.0	272.4	293	396.8
2	55	0	0.0	0.0	0	2.3	0	0.0	0.0	0	0.0	0	0.0	0.0	0	0.2
3	55	40828	41104.0	41110.9	41303	227.5	40678	**40853.5**	**40835.6**	**40927**	511.8	**40640**	40953.5	40961.2	41138	460.2
4	55	15091	15576.0	15621.3	16423	221.2	**14653**	**14653.0**	**14655.7**	**14675**	497.1	**14653**	14870.5	14854.9	15060	320.8
5	55	0	**0.0**	2.1	12	100.9	0	**0.0**	**0.7**	**3**	219.8	0	5.0	4.8	11	414.6
6	55	0	0.0	0.0	0	2.1	0	0.0	0.0	0	0.0	0	0.0	0.0	0	0.2
7	55	36489	37357.5	37308.6	37973	163.2	**35883**	**35976.0**	**35972.1**	**36066**	557.0	35979	36234.0	36235.6	36424	380.0
8	55	20624	21417.0	21386.7	22457	236.3	**19871**	**19871.0**	**19871.4**	**19880**	541.8	**19871**	19984.0	19980.9	20117	296.8
1	65	**295**	**317.5**	**319.1**	**350**	354.5	333	358.5	355.7	368	460.8	331	365.0	361.3	391	701.9
2	65	0	0.0	0.0	0	4.0	0	0.0	0.0	0	0.0	0	0.0	0.0	0	0.3
3	65	**57779**	58249.5	58266.8	58653	440.7	57880	**58122.0**	**58097.1**	**58276**	1023.0	57947	58276.5	58288.3	58590	911.5
4	65	34468	35399.0	35365.4	36107	466.7	**34410**	**34535.0**	**34522.5**	**34628**	939.7	34457	34898.0	34874.5	35138	647.0
5	65	**13**	**24.5**	**25.3**	**38**	347.5	30	35.5	35.0	41	521.2	28	37.5	36.7	42	710.0
6	65	0	0.0	0.0	0	4.0	0	0.0	0.0	0	0.0	0	0.0	0.0	0	0.3
7	65	56246	57037.5	57027.5	57825	352.7	55355	**55488.5**	**55473.6**	**55612**	1143.4	**55331**	55718.0	55676.6	56075	718.1
8	65	29308	30099.5	30155.9	31074	349.9	**27114**	**27115.5**	**27130.6**	**27164**	1029.9	27164	27393.5	27428.9	27715	601.1
1	75	**63**	**313.0**	**311.6**	368	610.2	317	334.0	334.1	**347**	811.1	287	335.0	331.2	356	1225.9
2	75	0	0.0	0.0	0	6.6	0	0.0	0.0	0	0.1	0	0.0	0.0	0	0.4
3	75	**78211**	**78541.5**	**78604.1**	79088	738.9	**78211**	78691.0	78689.4	**78859**	1875.0	78330	78708.5	78697.7	79162	1601.2
4	75	35826	37592.0	37514.3	38333	537.1	**35323**	**35433.0**	**35413.6**	**35487**	1671.1	35335	36056.5	35986.0	36280	1040.5
5	75	0	0.0	0.0	0	7.0	0	0.0	0.0	0	0.0	0	0.0	0.0	0	15.1
6	75	0	0.0	0.0	0	7.9	0	0.0	0.0	0	0.0	0	0.0	0.0	0	0.7
7	75	61513	62201.0	62216.8	63284	564.6	**60217**	**60481.0**	**60452.6**	**60556**	1848.9	60332	60730.5	60701.9	61014	1187.9
8	75	40277	42271.0	42018.5	42964	719.7	**38368**	**38453.0**	**38456.8**	**38548**	2000.3	38426	39025.0	39033.2	39333	996.5
1	85	**453**	**515.5**	**511.0**	**557**	883.8	531	563.0	559.2	579	1355.5	519	545.0	547.9	599	1990.8
2	85	0	0.0	0.0	0	10.6	0	0.0	0.0	0	0.0	0	0.0	0.0	0	0.6
3	85	98540	**98957.0**	**98949.0**	**99250**	1075.8	98794	99122.5	99118.6	99296	3022.4	**98389**	99058.5	98969.0	99524	2684.9
4	85	80693	81785.5	81702.6	82728	1301.3	**80338**	**80731.5**	**80695.5**	**80962**	2832.4	80797	81283.0	81330.0	82031	1778.3
5	85	**333**	**374.5**	**373.5**	**409**	971.0	393	418.0	417.5	436	1400.8	375	415.0	409.7	443	1986.2
6	85	0	0.0	0.0	0	10.7	0	0.0	0.0	0	0.1	0	0.0	0.0	0	0.6
7	85	89654	90574.5	90569.2	91447	905.6	**88089**	**88441.0**	**88402.1**	**88598**	3217.0	88130	88624.5	88585.4	88984	2046.4
8	85	77919	79368.5	79299.5	80612	1057.8	**75217**	**75424.0**	**75401.9**	**75517**	3714.4	75317	76212.0	76206.1	76794	1637.6

The computational results in Tables 4.6 and 4.7 show that GRASP+ONS is competitive to the ACO and GRASP algorithms with respect to the best solutions found. In terms of median and worst solutions, GRASP outperformed GRASP+ONS marginally. The path relinking method employed in the GRASP+ONS is also shown to be efficient in improving solution quality. We also found that for problem instances with low processing time variances, low tardiness factors, and narrow due date ranges, GRASP+ONS outperformed GRASP in terms of best, median, and worst solutions.

4.6. Concluding Remarks

In this chapter, we presented a new problem independent algorithm, named ONS algorithm, for single machine scheduling problems. As a general purpose and problem independent algorithm, the ONS algorithm is also applicable to solving a wide variety of problems whose solutions can be represented with permutations. The performance of the ONS algorithm was evaluated based on SMSP with unequal release dates and SMSP with sequence dependent setup times. The computational results show that the ONS algorithm is efficient in solving these single machine

Chapter 5 Tabu Search Algorithms for the Open Shop and Routing Open Shop Scheduling Problems

In this chapter, fast tabu search meta-heuristics for the open shop scheduling problem (OSSP) and the routing open shop scheduling problem (ROSSP) are presented. One new neighborhood is proposed for the OSSP and two new neighborhoods are proposed for the ROSSP. Moreover, an exact feasibility checking method is developed to remove infeasible moves quickly. To test the performance of the proposed tabu search algorithms, comprehensive computational experiments were carried out. The computational results show that the algorithms proposed in this chapter are able to find high-quality solutions within reasonable computation times.

5.1 Introduction

The OSSP can be described as follows. There are a set of jobs and the operations of each job have to be processed on different machines without restrictions to the operation processing order. In addition, each machine can only process, at most, one operation at a time and the operation cannot be interrupted once it is started. This problem is also called a non-preemptive OSSP. The OSSP whose objective is minimizing makespan is denoted as $O \| C_{\max}$ according to the classification of Graham et al. (1979). A schedule of an OSSP is an assignment of the operations with the operation processing order on each machine and the processing order of the operations belonging to the same job.

The OSSP is similar to the job shop scheduling problem (JSSP) with the exception that there is no processing order restriction placed on operations that belong to the same job. As a result, the OSSP has a larger solution space compared to the

JSSP. It was shown by Garey and Johnson (1979) that the general OSSP is an NP-hard problem. It was also proved by Gonzalez and Sahni (1976) that the OSSP with $m = 3$ is NP-complete, where m is the number of machines. However, it has been shown that some specially structured OSSPs with $m \geq 3$ are polynomially solvable (Fiala 1983).

By considering the transportation or setup times, the ROSSP becomes an extension of the OSSP, and is denoted as $RO \,||\, C_{\max}$. It is assumed in the ROSSP that machines are initially located at the same node (depot node) and have to travel along the transportation network to process the jobs and will return to the depot after all operations are processed. The transportation or setup times can be symmetric or asymmetric. In this study, we only consider the symmetric transportation times and the objective of the ROSSP is to minimize the makespan. It is noted that the transportation times and setup times can be handled in the same way in the ROSSP.

5.2 Problem and Schedule Formulation

In this section, we give a formal definition of the OSSP and the ROSSP followed by the definition of schedules.

The following notations are used throughout this chapter.

M_i	The machine on which operation i is processed
J_i	The job to which operation i belongs
$PM(i)$	The predecessor(s) of operation i on machine M_i in a schedule
$PJ(i)$	The predecessor(s) of operation i belonging to job J_i in a schedule
d_i	The processing time of operation i
p_{ij}	The processing time of an operation that belongs to job J_i and has to be processed on machine M_j
B_i	the i^{th} block on the critical path

5.2.1 Disjunctive Graph Problem Representation

For an OSSP or ROSSP, there are a set of jobs $J = \{J_1, ..., J_n\}$, and a set of machines $M = \{M_1, ..., M_m\}$. We assume that each job has *m* operations and all the operations of any job have to be processed on different machines. The first assumption can be relaxed because when a job has less than *m* operations, dummy operations with zero processing time can be added. The processing time of operation *i* is denoted as p_i. The OSSP and the ROSSP are generally modeled using a disjunctive graph, which was originally proposed by Roy and Sussmann (1964) for the JSSP. In the disjunctive graph *G* for the OSSP or ROSSP, the nodes, which correspond to the operations, are numbered from 1 to *N*, where $N = m \times n$ is the total number of operations. Any two operations that belong to the same job are connected to each other by two disjunctive arcs that go in opposite directions. Any two operations that have to be processed on the same machine are also connected to each other by two disjunctive arcs that go in opposite directions. The oppositely directed arcs between two operations denote that at most one operation can be processed at a time on a machine or at most one operation of a job can be processed at a time. The disjunctive arcs in graph *G* form $m + n$ cliques. *m* cliques correspond to *m* machine and *n* cliques correspond to *n* job. In graph theory, a clique is defined as a graph where any two nodes are connected with each other. For an OSSP, all arcs emanating from a node each have a length equal to the processing time of the source operation that is represented by the node. For a ROSSP, all arcs emanating from a node each have a length equal to the sum of the processing time of the source operation that is represented by the node and the transportation time from the source operation to this node. In addition, there are two dummy nodes with zero processing time, nodes 0 and $N+1$, representing the source node and the sink node respectively. The source node has a conjunctive arc sinking into each of the *N*

operations. The sink node has a conjunctive arc emanating from each of the N operations. A disjunctive graph of an OSSP with $n = m = 3$ is illustrated in Figure 5.1, in which the solid lines denote the conjunctive arcs and the dotted lines denote disjunctive arcs. For the example in Figure 5.1, operations 1, 2 and 3 belong to J_1, operations 4, 5 and 6 belong to J_2, operations 7, 8 and 9 belong to J_3, operations 1, 4 and 7 have to be processed on machine M_1, operations 2, 5 and 8 have to be processed on machine M_2, and operations 3, 6 and 9 have to be processed on M_3.

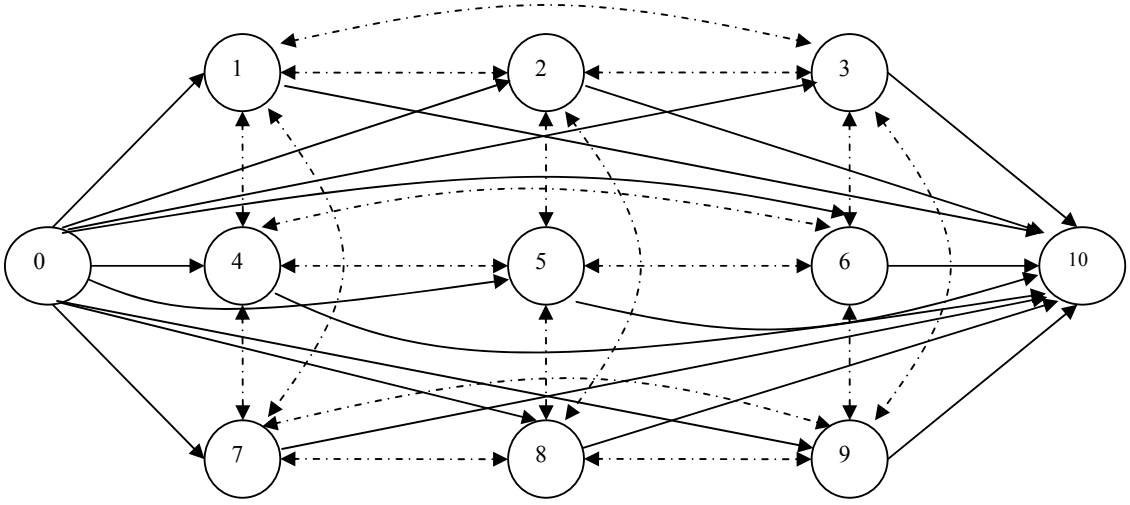


Figure 5.1 An example of a disjunctive graph for an OSSP

5.2.2 Acyclic Graph Schedule Representation

A schedule of an OSSP or ROSSP is an assignment of the operations with the operation processing order on each machine and the processing order of the operations belonging to the same job. Finding a feasible schedule is equivalent to selecting arcs from all the oppositely directed arcs, which means that the two operations on each end of the arc must be processed in the order either preceding or succeeding the other one.

As stated by Brucker et al. (1997), a selection S defines a feasible schedule if and only if,

(1) all disjunctive arcs are fixed, that is, there is no dotted line in the graph, and

(2) the resulting graph $G(S)$ is acyclic.

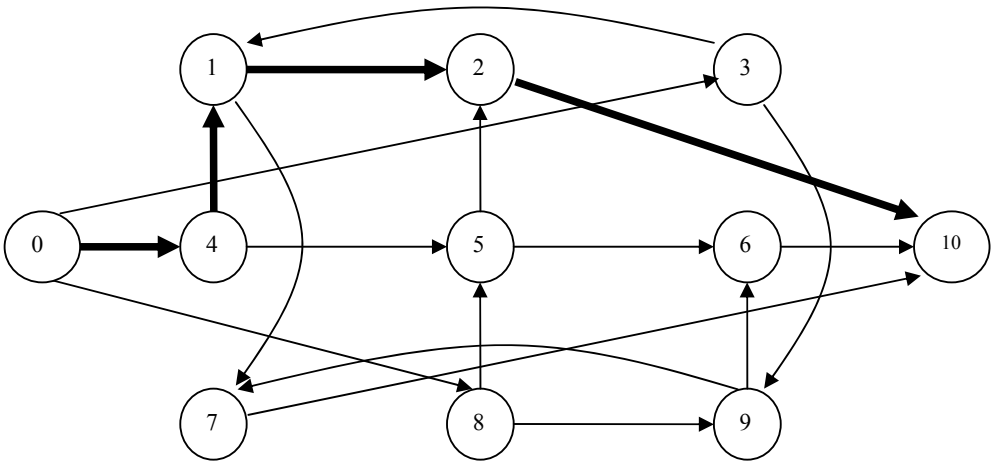


Figure 5.2 Illustration of a feasible schedule

An acyclic graph can be constructed easily based on a feasible schedule. In an acyclic graph, each node has at most two source nodes and two sink nodes. This is because each operation can have at most one immediate preceding operation that is processed on the same machine and at most one immediate preceding operation that belongs to the same job. Correspondingly, each operation can have at most one immediate succeeding operation that is processed on the same machine and at most one immediate succeeding operation that belongs to the same job. Topological sorting of a network is to sort the nodes in a network into topological order in which no node appears in it until after all nodes appearing on all paths leading to the particular node have been listed. Given a feasible schedule, the longest path in the corresponding acyclic graph is defined as the critical path and the length of the critical path is called

the makespan, which is equal to the maximum completion time among all the $N+2$ operations. The critical path and the makespan of a schedule can be obtained based on the following steps proposed by Christofides (1975).

Step 1. Topological sorting of the acyclic graph. Topological sorting can be carried out by the labeling algorithm proposed by Kahn (1962). In a topologically sorted acyclic graph, node *i* is always sorted prior to *j* if arc (i, j) exists;

Step 2. Determine the heads and tails of all nodes in the acyclic graph. The heads and tails are initially set to zero.

$$h_i = \max\left\{h_{IPM(i)} + d_{IPM(i)}, h_{IPJ(i)} + d_{IPJ(i)}\right\},$$

$$t_i = \max\left\{t_{ISM(i)} + d_{ISM(i)}, t_{ISJ(i)} + d_{ISJ(i)}\right\};$$

Step 3.To find a critical path, we need to track backwards from the sink of the acyclic graph towards the source following the critical nodes. If more than one critical node exists, select one arbitrarily.

Topological sorting is to sort the nodes in a network in topological order; no node appears in the sorted node list until all nodes leading to the particular node have been listed. An acyclic graph with a critical path (bold line) for a feasible schedule is illustrated in Figure 5.2. The makespan of the schedule or the acyclic graph is given by $C_{\max} = t_0 = h_{N+1}$. Node *i* is on a critical path or called a critical node if $h_i + d_i + t_i = C_{\max}$. A critical path can be decomposed into blocks. Here a block is defined as a chain of successive operations on a critical path that have to be processed on the same machines, or that belong to the same job. It is noted that there are at least two operations in any block. For example, the critical path in Figure 5.2 is $0 \to 4 \to 1 \to 2 \to 10$, which can

be decomposed into blocks $\{4 \to 1\}$ and $\{1 \to 2\}$. Operations 1 and 4 have to be processed on machine M_1 and operations 1 and 2 belong to job J_1.

5.3 Feasibility Checking Procedure

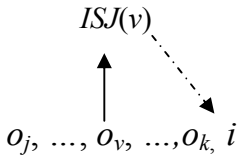
When more than one move is carried out simultaneously, or a move that is not along the critical path is carried out, it is possible to create a directed cycle in the graph, which means that the corresponding solution obtained is infeasible. To perform a feasibility test, the standard labeling algorithm described in Kahn (1962) can be used. As this procedure is computationally expensive, Dell'Amico and Trubian (1993) presented an estimated method to test the operation move feasibility. The basic idea of their feasibility checking method is that there cannot exist a path from *i* to *j* if $h_i + d_i > h_j$ holds. This method is widely used by many researchers who have worked on the JSSP and the OSSP. However, applying this method to ensure the feasibility comes at the expense of omitting a few feasible solutions, as pointed out by Dell'Amico and Trubian (1993). In this subsection, we present an exact method for the feasibility test. This method is also applicable to the JSSP.

We first describe four lemmas before the details of the feasibility checking method are presented.

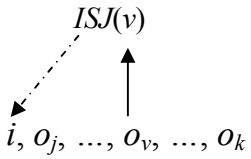
Lemma 5.1: In a feasible solution, for a move of an operation *i* to the first position for the successive operation sequence $(o_j, ..., o_k, i)$ to produce the successive operation sequence $(i, o_j, ..., o_k)$ on the same machine, a cycle in the resulting graph exists if and only if there is a path from at least one of the operations $ISJ(v)$ to operation *i*, where $v \in \{o_j, ..., o_k\}$.

Proof:

We assume there is a path from operation $ISJ(v)$ to *i*, that is, the following successive operation processing sequence exists.



The solid line indicates the immediate succeeding relationship and the dotted line indicates the succeeding operation relationship respectively. After moving operation *i* to the first position of the block, we have the following new processing sequence of operations.



It is indicated that there is a cycle in the new sequence that results in an infeasible schedule, thus completing the proof.

Lemma 5.2: In a feasible solution, for a move of operation *i* to the last position for the successive operation sequence $(i, o_j, ..., o_k)$ to produce the successive operation sequence $(o_j, ..., o_k, i)$ on the same machine, a cycle in the resulting graph exists if and only if there is a path from operation *i* to at least one of the operations $IPJ(v)$, where $v \in \{o_j, ..., o_k\}$.

Proof: Refer to the proof of Lemma 5.1.

Lemma 5.3: In a feasible solution, for a move of operation *i* to the first position for the successive operation sequence $(o_j, ..., o_k, i)$ to produce the successive operation sequence $(i, o_j, ..., o_k)$ belonging to the same job, a cycle in the resulting graph exists if

and only if there is a path from at least one of the operations $ISM(v)$ to operation i, where $v \in \{o_j, ..., o_k\}$.

Proof: Refer to the proof of Lemma 5.1.

Lemma 5.4: In a feasible solution, for a move of operation i to the last position for the successive operation sequence $(i, o_j, ..., o_k)$ to produce the successive operation sequence $(o_j, ..., o_k, i)$ belonging to the same job, a cycle in the resulting graph exists if and only if there is a path from operation i to at least one of the operations $IPM(v)$, where $v \in \{o_j, ..., o_k\}$.

Proof: Refer to the proof of Lemma 5.1.

To check whether there is a path from a node to another node, we use a precedence matrix to record the precedence relationships. In our implementation, a square matrix whose dimension is equal to the number of operations is used to record the precedence relationships. The precedence matrix not only records the immediate precedence relationships but also the non-immediate precedence relationships. In an acyclic graph, the value of cell(i, j) of the precedence matrix is set to TRUE if node i is a predecessor of node j in the corresponding topological acyclic graph.

The procedure to build the precedence matrix for a topologically sorted acyclic graph is as follows:

Step 1: Initialize all the cell values of the precedence matrix to FALSE;

Step 2: Starting from node $i = 1$, find the immediate successors $ISM(i)$ and $ISJ(i)$ of node i, set the cell values $(i, ISM(i))$:= TRUE and $(i, ISJ(i))$:= TRUE;

Step 3: Find the predecessors $PM(i)$ and $PJ(i)$ of node i, set the cell values $(PM(i), i)$:= TRUE and $(PJ(i), i)$:= TRUE;.

Step 4: Set $i := i + 1$, if $i < N$, go to Step 2; otherwise, stop.

Based on the precedence matrix, it is easy to check whether there exists a path from a node to another node.

After performing a move of operations, a topological sorting should be carried out. Since the nodes affected are limited in proximity within the old topological sorting, topological sorting is applied only to those nodes that are affected and hence most of the old topological sorting can be kept. Accordingly, the precedence matrix needs only to be updated partially based on the change of the old topological sorting of the graph. This strategy can reduce the computation time significantly.

5.4 Tabu Search Strategies

Tabu search is an iterative improvement approach designed for optimization problems. Tabu search (TS) was initially proposed by Glover (1986); it has now become one of the most efficient meta-heuristics for solving combinatorial optimization problems. The basic idea of TS is of using short-term memory to record recent moves in the search to prevent the search from returning to a previously visited neighbor. The short-term memory is also called the tabu list. Moreover, a long-term memory for diversification purpose is applied to ensure that the search will not be restricted to a small neighborhood. Recording the recent moves in the tabu list does not mean recording the whole or part of the solution in memory. On the contrary, the tabu list only memorizes the attributes of the moves. Under certain circumstances, the memory may forbid some moves that may lead to an improvement of the solution. In this case, an aspiration criterion is introduced which is used to disable the memory function temporarily. Tabu search prevents cycling and guides the search toward unexplored regions by forbidding solutions with certain attributes.

As TS has been shown to be one of the most efficient meta-heuristics for solving difficult combinatorial optimization problems (Taillard and Parallel 1994, Nowicki, and Smutnicki 1996, Liaw 1999), we use TS as a platform to evaluate the efficiency of the neighborhoods and search strategies proposed in this research work. The tabu search strategies used in this study are given in the following sub-sections.

5.4.1 Aspiration Criterion

Under certain circumstances, the tabu search may forbid a move that may generate an improved solution. Therefore, an aspiration criterion is introduced to disable temporarily the tabu status of the prospective move so that the move is allowable. The aspiration criterion in the proposed tabu search is as follows: a move is allowed only if the estimated makespan is less than the makespan of the best solution found so far.

It may happen that at certain iterations, all possible moves are forbidden and none of the moves satisfies the aspiration criterion. In this case, we follow the strategy described in Glover (1989) to remove the oldest tabu in the tabu list and additionally replicate the most recent tabu and add it to the tabu list until the moves selected are allowable.

5.4.2 Back Jump Tracking

Back jump tracking strategy was proposed by Nowicki and Smutnicki (1996). Unlike the multi-start strategy, which starts from different initial solutions whenever the best solution cannot be improved within a fixed number of iterations, the back jump tracking strategy records a certain number of best solutions found during previous iterations. When the best solution is not improved for a fixed number of

iterations, a solution stored previously is used as the initial solution to restart the search. In addition, to prevent repetition of the same search history, the tabu list associated with the solution generated in the next iteration at which the best solution is found is also recorded. This strategy is able to use the information found during the previous runs and always does intensive search to explore the neighbors of the best solutions found at different iterations. Nevertheless, in the case that the initial solution is a good solution and cannot be improved within a certain number of iterations, the search will stop after a certain number of iterations without implementing back jump tracking. To overcome this disadvantage, we propose to store some solutions in the elite solution list before the tabu search starts. This strategy is able to overcome the disadvantages of both the multi-start strategy and the original back jump tracking strategy.

The modified back jump tracking strategy in TS is implemented as follows. A *maxb* number of solutions are stored in the elite solution list *B*, where *maxb* is a fixed number. If a best solution s' is found at iteration *k*, the updated tabu list at iteration $k+1$ is stored into the elite solution list *B* together with the solution s'. If the total number of solutions in *B* is more than *maxb*, the oldest solution is removed from the list. Whenever the number of iterations that have no improvements is reached, the most recent solution in the elite solution list *B* is used as the new starting solution to restart the search and is removed from the elite solution list *B* simultaneously. The iteration counter is reset to zero whenever the search starts from a new solution in the elite solution list *B*. Each single back jump tracking search stops when the total number of iterations performed exceeds the maximum number of iterations, *maxiter*. The whole back jump tracking process stops when either the elite solution list *B* is empty or the

optimal solution is found and proved. The elite solutions list *B* is implemented as a first in last out (FILO) list.

5.4.3 Cycle Detection Method

Cycle detection strategy is used to reduce the fruitless cycle search when it exists but does not affect the final solution quality. We adopt the cycle detection strategy used by Nowicki and Smutnicki (1993). In a cycle situation, a solution after iteration k is similar to the solution after $k+\Delta, k+2\times\Delta$, $k+3\times\Delta,\ldots$, where Δ is the length of the cycle. In order to detect a cycle, the makespans of a period length of iterations, $\delta_{\max}$, which must be greater or equal to Δ, is recorded. If there is a period δ, where $1 \le \delta \le \delta_{\max}$, and $C_{\max}^{k-\Delta} = C_{\max}^{k}$ is true for $k = iter + 1 - \delta$, where $C_{\max}^{k}$ is the makespan found at iteration *k*, a cycle is detected.

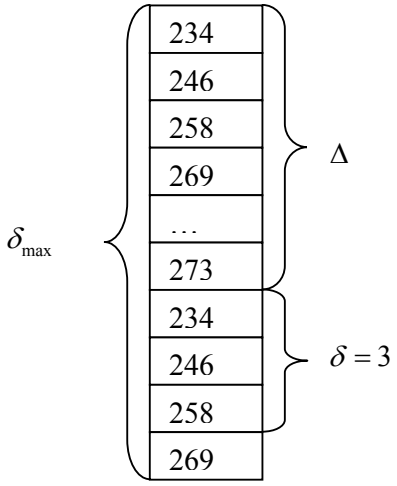


Figure 5.3 An illustration of recorded makespans for cycle detection

The cycle detection mechanism described above is illustrated in Figure 5.3. With a larger value of $\delta_{\max}$, the mechanism is able to detect a longer cycle but at the expense of higher memory needed to record the makespan list and longer computation time. A

larger value of δ increases the confidence level of the cycle detection results but may increase the number of unnecessary comparisons of makespans. The values of δ and $\max \delta$ are set experimentally as a compromise between the amount of computation required and the confidence level of the cycle detection result. It is noted that the values of δ and $\delta_{\max}$ are dependent on the problem characteristics and other tabu search parameters, i.e. the range of objective values of the solution space, the move attributes, and the length of the tabu list. Our computational experiments show that setting $3 \le \delta \le 5$ is enough to detect the cycle with less than 1% occurrence of false alarms.

5.5 Application of TS to the Open Shop Scheduling Problem

In this section, we apply the tabu search algorithm to the OSSP using the search strategies discussed in Section 5.4. The objective of the problem $O_m \,||\, C_{\max}$ is to minimize the makespan.

5.5.1 Initial Solutions

We apply the dispatching rule, DS/LTRP rule, described in Liaw (1999), to generate the initial solutions. The dispatching rule is described as follows. When more than one machine is idle at the same time, select the machine that has the longest total remaining processing time; otherwise, choose the first idle machine and choose the operation belonging to the job that has the longest total remaining processing time on the other machines. If no such operation exists, the machine remains idle until an operation has been completed on some other machine such that an operation is ready to be processed by the machine that was chosen previously. As stated in Section 5.4, initial solutions are stored in the elite solution list *B* before the tabu search starts. Here,

we prefer to store different initial solutions in the elite solution list B to prevent repetition. However, the DS/LTRP rule is a deterministic procedure and can only generate one unique solution. To overcome this drawback, we implement a greedy randomized procedure to generate different initial solutions. Whenever more than one job is available, we select the next job to be scheduled using a greedy cost function as stated in Armentano and Araujo (2006). A restricted candidate list (RCL) is formed to select the operation to be scheduled next. Let $R_{\max}$ and $R_{\min}$ denote the maximum and minimum total remaining processing time on the other machines respectively for the jobs whose operations are available on the selected machine. The number of jobs in the RCL is limited by using a threshold parameter $\alpha \in [0,1]$. As we want to select the job with the longest remaining processing time, all jobs with costs larger or equal to $R_{\max} - \alpha \cdot (R_{\max} - R_{\min})$ are included in the RCL. A job is selected randomly from the RCL using a uniform distribution and the available operation belonging to the selected job is scheduled next. Thus, $\alpha = 0$ corresponds to the greedy choice and $\alpha = 1$ results in a pure randomized job selection. Therefore, the threshold parameter α controls the balance between the greedy and the randomized solution construction.

5.5.2 Lower Bound

The OSSP lower bound is computed using the method proposed by Pinedo (2002):

$$C_{\max}^{LB} = \max\left\{\max_{j}\left\{\sum_{i=1}^{n} p_{ij}\right\}, \max_{i}\left\{\sum_{j=1}^{m} p_{ij}\right\}\right\}. \tag{5.1}$$

5.5.3 Neighborhoods

Neighborhood N_1:

Liaw (1999) defined a neighborhood structure for the OSSP. The neighborhood defined by Liaw (1999) is called neighborhood N_1 in this thesis. N_1 considers the reversal of arc (i, j) in a block where either node i is the first node or j is the last operation in the block. These arcs are called candidate arcs. For each candidate arc (i, j), the following arc reversals are considered if $M_i = M_j$,

(1) Arcs (i, j);

(2) Arcs (i, j) and $(IPJ(j), j)$;

(3) Arcs (i, j) and $(i, ISJ(i))$;

(4) Arcs (i, j), $(IPJ(j), j)$ and $(i, ISJ(i))$.

Similarly, for an candidate arc (i, j) with $J_i = J_j$, the following arc reversals are considered

(5) Arcs (i, j)

(6) Arcs (i, j) and $(IPM(j), j)$

(7) Arcs (i, j) and $(i, ISM(i))$

(8) Arcs (i, j) $(IPM(j), j)$ and $(i, ISM(i))$.

However, as pointed out in Liaw (1999), it may happen for N_1 that at certain iterations, all feasible moves are forbidden and no move satisfies the aspiration criterion. The reason for this is that the size of the neighborhood N_1 is too small. In this study, we propose a neighborhood N_2 that is able to circumvent the disadvantage of neighborhood N_1.

Neighborhood N_2:

Let operation v be an operation in block $B = \{b', v, b''\}$, where v denotes an operation whereas b' and b'' denote blocks having at least one operation.

Neighborhood N_2 is defined by moving v to the first or last position in block B and relocating v to the appropriate position on the non-critical path at the same time.

The neighborhood N_2, which concentrates on moving an operation within a block to the first or last position of the block, has a larger size than N_1. Moving v to the first or last position in block B obtains two schedules, $\left(v, b', b''\right)$ and $\left(b', b'', v\right)$. In addition, relocating operation v to some position on the non-critical path is carried out simultaneously. Let us assume that the operations in block B have to be processed on the same machine and the first operation in block b' is operation w and hence we have $h_w = h_{IPJ(w)} + d_{IPJ(w)} \geq h_{IPM(w)} + d_{IPM(w)}$. When an operation v is moved to the first position in block B, the head of operation v is determined by $h_v' = \max\left\{h_{IPM(w)} + d_{IPM(w)}, h'_{IPJ(v)} + d'_{IPJ(v)}\right\}$, where h_v' stands for the head of operation v after it is moved to the new position.

Two cases will be considered in order to relocate v onto the non-critical path.

Case 1: $h_{IPM(w)} + d_{IPM(w)} < h_{IPJ(v)} + d_{IPJ(v)}$

Relocating operation v to a position in the operation sequence of job J_v such that $h_{IPM(w)} + d_{IPM(w)} < h'_{IPJ(v)} + d'_{IPJ(v)} < h_{IPJ(v)} + d_{IPJ(v)}$. It is noted that there is no need to move operation v to a position such that $h_{IPM(w)} + d_{IPM(w)} \geq h'_{IPJ(v)} + d'_{IPJ(v)}$, as the head of operation v is determined by the maximum value of $h_{IPM(w)} + d_{IPM(w)}$ and $h'_{IPJ(v)} + d'_{IPJ(v)}$.

Case 2: $h_{IPM(w)} + d_{IPM(w)} \geq h_{IPJ(v)} + d_{IPJ(v)}$

Do nothing as relocating operation v to a position in the operation sequence of job J_v cannot reduce h_v'.

For Case 1, it is possible that there is more than one position where operation v can be moved to. In this circumstance, all the possible positions should be tested. Similarly, moving an operation to the last position in a block is also considered in neighborhood N_2.

To estimate the effect of the move of operations for neighborhoods N_1 and N_2, we use the approach similar to that proposed by Dell' Amico and Trubian (1993) for the JSSP. For each new possible solution, its estimated makespan is computed by the heads and tails of those operations whose preceding or succeeding (including the non-immediate predecessors and the non-immediate successors) operations are changed. The maximum value of $h_i + d_i + t_i$ is the estimated makespan. The estimated makespan provides a valid lower bound for the exact makespan of the new solution. Once a new solution is accepted, its exact makespan is calculated using the approach developed by Christofides (1975) as described in subsection 5.2.2.

5.5.4 Tabu Search Algorithm for the OSSP

Tabu list is used to memorize the moves and to prevent cycle traps. Each time a move is performed, the attributes related to the move are pushed back into the tabu list. The moves are forbidden only for a certain number of iterations in tabu search. This mechanism helps to prevent cycle traps after a deterioration of the objective value move has been accepted. However, it is critical to design the move attributes so that cycle traps can be prevented and the potentially good solution space is not forbidden from being searched at the same time.

For neighborhood N_1, we follow Liaw (1999) to memorize the reversal of all arcs involved. For neighborhood N_2, its immediate succeeding operations,

$ISM(i)$ and $ISJ(i)$ in the new solution are found and the move attributes $(i, ISM(i))$ and $(i, ISJ(i))$ are added to the tabu list for the forward move of an operation *i*. For a backward move of an operation *i*, its immediate preceding operations, $IPM(i)$ and $IPJ(i)$ in the new solution are found and the moves $(i, ISM(i))$ and $(i, ISJ(i))$ are added to the tabu list. If the immediate preceding operation or immediate succeeding operation is a dummy node, we will ignore the related attributes. A move is forbidden if at least one of the arcs involved is in the tabu list. It is noted that the move attributes defined for neighborhood N_2 is not as straightforward as those defined for neighborhood N_1. However, our computational experiments show that the move attributes defined here for neighborhood 2 N_2 is very effective in finding better solutions and preventing cycle traps.

Tabu list is implemented as a first-in-first-out list with a tabu length, which is the number of iterations a move is kept as tabu, and varies according the rules given below.

(1) If a new best solution is found, increase the length of the tabu list if its length is less than a threshold *maxTabuLength*.

(2) Each time a back jump tracking is triggered, decrease the length of the tabu list if its length is greater than a threshold value *minTabuLength*.

If the threshold value of *minTabuLength* is too large, the tabu list will forbid too many moves even when there are potentially better moves; on the other hand, using too small a value of *minTabuLength* can possibly cause the search to be trapped into cycles. The length of the tabu list should be set experimentally.

It is quite easy to implement a classical tabu search algorithm. We will start from a feasible initial solution. At each iteration, the best feasible non-tabu move is selected and a new solution is generated by carrying out the move. If the makespan of the new

solution is better than the best solution found so far, it is added to the elite solution list. The algorithm stops in either of the two cases: (1) a solution with makespan is equal to the lower bound of makespan is found, or (2) the elite solution list is empty and the number of iteration is greater than *maxIteration*.

The procedure of the tabu search algorithm for OSSP is given below.

Step 1. Set iter :=0, initialize *maxiter*, *maxb*, *max#NoImprov*., *minTabuLength*, *maxTabuLength*, compute the lower bound of makespan $C_{\max}^{LB}$;

Step 2. Construct *maxb* number of solutions based on the procedure described in subsection 5.5.1 with $\alpha > 0$;

Step 3. Construct an initial solution based on the procedure described in subsection 5.5.1 with $\alpha = 0$;

Step 4. Set iter:= iter + 1. If (iter > *maxiter*) OR (*#NoImprov*. > *max#NoImprov*.) go to Step 8; otherwise, find the best feasible move from all the possible moves of N_1 and N_2, compute the exact makespan $C_{\max}^{'}$ of the new solution $s^{'}$, update the tabu list;

Step 5. If (isCycle(iter, $C_{\max}^{'}$) = TRUE) go to Step 8; otherwise, go to Step 6;

Step 6. If ($C_{\max}^{'} < C_{\max}^{best}$) set $C_{\max}^{best} := C_{\max}^{'}$, go to Step 7; otherwise, go to Step 4;

Step 7. If ($C_{\max}^{'} = C_{\max}^{LB}$), Stop; otherwise push the solution $s^{'}$ into elite solution list *B*; go to Step 4;

Step 8. If the elite solution list *B* is empty, stop; otherwise, use the most recent solution in elite solution list *B* as the starting solution and remove the elite solution from list *B*, set iter := 0, go to Step 4.

In Step 1, the problem lower bound is computed using formula (5.1). In Step 5, the function isCycle(iter, $C_{\max}^{'}$) is used to detect whether there exists a cycle or not

based on the procedure described in subsection 5.4.3. If isCycle(iter, $C'_{\max}$) = TRUE, then the search procedure will jump to Step 8 to use the most recent solution in the elite solution list *B* as the new starting solution.

5.5.5 Computational Results

To test the performance of the tabu search algorithm proposed in this work, the TS algorithm was coded in C++ and implemented on a Pentium 4 personal computer with 2.6GHz CPU and 512MB RAM. The widely used benchmark problems described in Taillard (1993) are used to evaluate the performance of the algorithm proposed in this chapter. The set of problems contains 60 hard benchmark problem instances ranging from small problems with 16 operations to large problems with 400 operations. Moreover, these problems are all square problems, in which the number of jobs is equal to the number of machines. It was observed by Taillard (1993) that the square OSSPs are harder to be solved compared to other problems.

The parameters of the tabu search algorithm should be set experimentally to ensure a compromise between computation time and quality of final solution obtained. In our implementation, the parameters are set to the following values: maximum number of iterations *maxiter*: = 40,000, maximum number of elite solutions, *maxb* =10, maximum number of no improvement, *max#NoImprov*. = 20,000, the lower threshold value of tabu length *minTabuLength* $= \lceil (n+m)/2 \rceil$, and the upper threshold value of the tabu length *maxTabuLength* $= (n+m)$. The detailed computational results are presented and compared with the results given by Liaw (1999) in Table 5.1. The first column in Table 5.1 indicates the problem type with the number of jobs, the number of machines, and the problem instance replication number. The lower bound of makespan $C^{LB}_{\max}$, obtained from formula (5.1), is shown in the second column. The third column

shows the optimal solution makespan of the corresponding OSSP. The best makespan obtained and its deviation from the optimal solution makespan, and the computation time in seconds are presented for both the algorithm of Liaw (1999) and our tabu algorithm. The relative deviation from the optimum solution objective is defined as follows: $Dev.(\%) = ((Z_{best} - opt.)/opt.) \times 100\%$.

Table 5.1 Results for the Taillard's benchmark problems

Problem	$C_{\max}^{LB}$	opt .	*Liaw (1999)*			Tabu Search Algorithm		
			Z_{best}	*Dev.* (%)	Time[1]	Z_{best}	Dev. (%)	Time
Taillard 4×4_1	186	193	193*	0.00	0	193*	0.00	0.67
Taillard 4×4_2	229	236	236*	0.00	0	236*	0.00	2.17
Taillard 4×4_3	262	271	271*	0.00	0	271*	0.00	0.55
Taillard 4×4_4	245	250	250*	0.00	0	250*	0.00	0.61
Taillard 4×4_5	287	295	295*	0.00	8	295*	0.00	0.64
Taillard 4×4_6	185	189	189*	0.00	1	189*	0.00	2.00
Taillard 4×4_7	197	201	201*	0.00	0	201*	0.00	1.41
Taillard 4×4_8	212	217	217*	0.00	8	217*	0.00	1.44
Taillard 4×4_9	258	261	261*	0.00	0	261*	0.00	2.69
Taillard 4×4_10	213	217	217*	0.00	1	217*	0.00	1.44
Taillard 5×5_1	295	300	300*	0.00	2	300*	0.00	9.25
Taillard 5×5_2	255	262	262*	0.00	0	262*	0.00	9.70
Taillard 5×5_3	321	323	326	0.93	82	**323***	0.00	11.14
Taillard 5×5_4	306	310	310*	0.00	0	310*	0.00	9.64
Taillard 5×5_5	321	326	326*	0.00	14	326*	0.00	10.88
Taillard 5×5_6	307	312	312*	0.00	1	312*	0.00	12.69
Taillard 5×5_7	298	303	303*	0.00	2	303*	0.00	8.41
Taillard 5×5_8	292	300	300*	0.00	32	300*	0.00	12.25
Taillard 5×5_9	349	353	353*	0.00	16	353*	0.00	14.44
Taillard 5×5_10	321	326	326*	0.00	24	326*	0.00	6.88
Taillard 7×7_1	435	435	435*	0.00	21	435*	0.00	17.75
Taillard 7×7_2	443	443	447	0.90	55	**443***	0.00	1.09
Taillard 7×7_3	468	468	474	1.28	128	**471**	0.64	43.83
Taillard 7×7_4	463	463	463*	0.00	75	463*	0.00	13.78
Taillard 7×7_5	416	416	417	0.24	94	**416***	0.00	6.99
Taillard 7×7_6	451	451	459	1.77	87	**457**	1.33	37.78
Taillard 7×7_7	422	422	429	1.66	75	**425**	0.71	34.69
Taillard 7×7_8	424	424	424*	0.00	32	424*	0.00	5.39
Taillard 7×7_9	458	458	458*	0.00	22	458*	0.00	1.23
Taillard 7×7_10	398	398	398*	0.00	11	398*	0.00	1.83
Taillard 10×10_1	637	637	646	1.41	190	**643**	0.94	78.48
Taillard 10×10_2	588	588	588*	0.00	1	588*	0.00	6.55
Taillard 10×10_3	598	598	601	0.50	126	601	0.50	53.89
Taillard 10×10_4	577	577	577*	0.00	101	577*	0.00	0.13
Taillard 10×10_5	640	640	644	0.63	188	**640***	0.00	33.41
Taillard 10×10_6	538	538	538*	0.00	2	538*	0.00	0.66
Taillard 10×10_7	616	616	616*	0.00	23	616*	0.00	9.05
Taillard 10×10_8	595	595	595*	0.00	25	595*	0.00	55.85
Taillard 10×10_9	595	595	597	0.34	157	**595***	0.00	8.77
Taillard 10×10_10	596	596	596*	0.00	55	596*	0.00	5.63
Taillard 15×15_1	937	937	937*	0.00	1	937*	0.00	0.61
Taillard 15×15_2	918	918	920	0.22	303	**919**	0.11	134.35
Taillard 15×15_3	871	871	871*	0.00	9	871*	0.00	0.66
Taillard 15×15_4	934	934	934*	0.00	3	934*	0.00	0.48
Taillard 15×15_5	946	946	949	0.32	293	**948**	0.21	149.45
Taillard 15×15_6	933	933	933*	0.00	32	933*	0.00	6.66
Taillard 15×15_7	891	891	891*	0.00	299	891*	0.00	68.78
Taillard 15×15_8	893	893	893*	0.00	3	893*	0.00	0.52
Taillard 15×15_9	899	899	910	1.22	301	**905**	0.67	136.94
Taillard 15×15_10	902	902	906	0.44	263	**902***	0.00	75.92
Taillard 20×20_1	1155	1155	1155*	0.00	114	1155*	0.00	39.19
Taillard 20×20_2	1241	1241	**1246**	0.40	654	1249	0.64	201.36
Taillard 20×20_3	1257	1257	1257*	0.00	2	1257*	0.00	2.31
Taillard 20×20_4	1248	1248	1248*	0.00	71	1248*	0.00	10.33
Taillard 20×20_5	1256	1256	1256*	0.00	14	1256*	0.00	0.66
Taillard 20×20_6	1204	1204	1204*	0.00	31	1204*	0.00	5.31
Taillard 20×20_7	1294	1294	1298	0.31	400	**1294***	0.00	47.17
Taillard 20×20_8	1169	1169	1184	1.28	866	**1182**	1.11	138.47
Taillard 20×20_9	1289	1289	1289*	0.00	13	1289*	0.00	1.56
Taillard 20×20_10	1241	1241	1241*	0.00	14	1241*	0.00	0.42
Average				0.23			0.12	

1- Computation time reported in seconds on Pentium-133 PC

(An asterisk indicates that the solution found is optimal and boldface indicates better solutions found by our tabu search algorithm or the algorithm proposed by Liaw (1999))

The computational results in Table 5.1 show that the proposed tabu search algorithm found 50 optimal solutions among the 60 problem instances. For the small sized problem instances, e.g. Taillard's 4×4 problem instances, all the 10 optimal solutions can be found in a short time. For problem instances of Taillard 5×5, our tabu search algorithm obtained 10 optimal solutions while the algorithm proposed by Liaw (1999) only obtained 9 optimal solutions. For Taillard's 7×7, 10×10, 15×15 and 20×20 problem instances, our tabu search algorithms obtained 6 more optimal solutions than the algorithm proposed by Liaw (1999). In addition, our tabu search algorithm found 15 better solutions than the algorithm proposed by Liaw (1999). Where the average relative deviation from the optimal solution makespan is concerned, our tabu search algorithm obtained an average deviation 0.12% for the 60 problem instances while the algorithm proposed by Liaw (1999) only obtained an average of 0.23%. The improvement is not very significant because both methods can get very good solutions that are optimal or near optimal solutions. As the computational experiments were conducted on different platforms, the computation times can only be compared approximately. However, the computation time in column 9 indicates that the computation time of our tabu search algorithm is reasonable. By analyzing $C_{\max}^{LB}$, Z_{best} and the computation time in Table 5.1, we find that when the final value of Z_{best} is larger than $C_{\max}^{LB}$, the TS algorithm will normally spend longer computation times than for other problem instances of the same size; the gap is large especially when the size of the problem is large. This is because once the objective value is equal to $C_{\max}^{LB}$, meaning that the optimal solution is found and proved, the search procedure will stop. If the Z_{best} is larger than $C_{\max}^{LB}$, the TS algorithm will search for better solutions until the elite solution list B is empty.

5.6 Application of TS to the Routing Open Shop Scheduling Problem

Machine scheduling problems where machines travel between jobs located at different nodes were studied by Averbakh and Berman (1996) and Averbakh and Berman (1999) for the flow shop problem. A $\frac{6}{5}$-approximate algorithm was proposed by Averbakh et al. (2005) for a ROSSP with two machines on a 2-node network. Averbakh et al. (2006) proved that the ROSSP with two machines on a 2-node network with n jobs is NP-hard. The same authors also proposed a heuristic for the general ROSSP based on the conclusion obtained for the routing flow shop problem.

For a routing open shop scheduling problem, it is assumed that jobs are located at nodes of an undirected transportation network $G =< V, E >$ with a set of nodes V and a set of edges E. It is also assumed that the machines are located at the same node (depot), have to travel between jobs to process operations, and will return to the depot after all operations are processed. The routing open shop is denoted as $RO \,||\, C_{\max}$ in Averbakh et al. (2006). The following notations are used throughout this subsection.

T - the optimum objective value for the transport network of the problem, related to traveling salesman problem TSP(G)

$l_i := \sum_{j=1}^{n} p_{ji}$ is the load of machine M_i;

$p_j := \sum_{i=1}^{m} p_{ji}$ is the length of job J_j;

$p_j^{'} = p_j + 2 \times d_{0j}$ is the job length plus two times of the distance from the depot to job j.

$L := \max_i l_i$ is the maximum machine load

$P := \max_j p_j$ is the maximum job length

$P^{'} := \max_j p_j^{'}$

For ease of presentation and without loss of generality, we assume that $|V| = mn$ and there is only one job at each node. Therefore, J_j can be used to denote both job and

network node. Here J_0 is used to denote the depot node where all machines are initially located and we let J_0 denote a job with m operations with zero processing. In the undirected transportation network $G =< V, E >$, the distance between different jobs can be represented by a $n \times n$ symmetric matrix, which is similar to the distance matrix in a symmetric traveling salesman problem. Adding the two matrices and considering the depot node J_0, a $(n+1)\times(n+1)$ asymmetric distance matrix M^i can be formulated for each machine M_i.

5.6.1 Initial Solutions

Every time a machine is available, select an operation with the earliest start time from all the operations available on the selected machine. When more than one machine is available, select the machine with the maximum remaining loads.

The steps for generating initial solutions are summarized below.

Step 1. Select an available machine. If more than one machine is available, select the machine with the largest remaining loads;

Step 2. Select an available operation to be processed on the machine, and then assign it to the machines. If all operations have been assigned, go to Step 4;

Step 3. If no operation is available, wait until one machine is available. Go to Step 1;

Step 4. Stop.

5.6.2 Lower Bound

Averbakh et al. (2006) proposed a lower bound $\max\{T+L, P\}$ for the 2-machine ROSSP. For the *m*-machine case, the value $\max\{\max\{L, P\}, T\}$ was used as the lower bound. We proposed a lower bound that is tighter than the lower bound proposed by Averbakh et al. (2006) as follows.

$$C_{\max}^{LB} = \max\left\{T + L, P^{'}\right\} \text{ for the } m\text{-machine ROSSP.} \quad (5.2)$$

The proof is straightforward. For any machine, the optimal tour for the TSP is independent of the job processing time. Therefore, $T + L$ is a lower bound of completion for the machine with the maximum load. For any job, one machine has to travel to the location of the job to process its first operation. If an operation of the job J_j is the first operation to be processed on the machine, the traveling distance is d_{0j}; otherwise, the distance is longer than d_{0j} based on the triangle inequality property. The machine has to travel back to the depot position after its last operation is processed. As all machines are located at the same depot node, and will return to the same depot node after all operations are processed, $p_j^{'}$ is a valid lower bound of time to complete job j. Therefore, $\max\left\{T + L, P^{'}\right\}$ is a valid lower bound for the ROSSP.

5.6.3 Neighborhoods

Neighborhood N_1 is adapted from the neighborhood structure defined by Liaw (1999) for the OSSP. As the neighborhood defined by Liaw (1999) does not consider the transportation or setup times, neighborhood N_1 for the ROSSP is extended from the neighborhood N_1 for the OSSP by taking the transportation or setup times into account when makespan is estimated for the ROSSP. Neighborhood N_2 for the ROSSP is extended from the neighborhood N_2 for the OSSP described in subsection 5.5.3 by considering the transportation or setup times. For the ROSSP, the makespan is composed of three components of time, namely operation processing time, waiting time, and machine traveling time. Therefore, it is possible to change the operation processing order inside a block to reduce the machine traveling time for those operations that require the same machine. Neighborhood N_3 is defined by changing of the operation processing order within a block where the operations have to be

processed on the same machine. In this work, adjacent pairwise interchange (API) of operations is implemented for the neighborhood N_3.

Lemma 5.5: For two adjacent operations which are inside of a block (but neither of the two operations is the first or the last operation of the block), and have to be processed by the same machine, the makespan of the ROSSP cannot be reduced if the transportation or setup times cannot be reduced after pairwise interchange.

Proof: If the transportation or setup times are ignored, the ROSSP is reduced to OSSP. For the OSSP, the adjacent pairwise interchange of operations within a block will not reduce the makespan (Mattfeld 1996). When the transportation or setup times are considered, the increase of transportation or setup times by adjacent pairwise interchange of operations will also increase the makespan of the ROSSP.

5.6.4 Tabu Search Algorithm

The same aspiration criterion, back jump tracking method and cycle detection method described in Section 5.4 are used for the tabu search algorithm developed for the ROSSP. For neighborhood N_1, the reversal of all arcs involved are stored in the tabu list. For neighborhood N_2, the arcs $(i, ISM(i))$ and $(i, ISJ(i))$ in the new solution are added to the tabu list. As for the neighborhood N_3, the arc reversed by adjacent pairwise interchange is recorded. The detailed implementation of the tabu search algorithm for the ROSSP is the same as that for the algorithm developed for the OSSP described in subsection 5.5.4.

5.6.5 Computational Results

Random problem instances are generated to test the performance of the tabu search algorithm developed for the ROSSP. Each problem instance is generated with four parameters: number of jobs, number of machines, the distribution of operation processing time, and the distribution of the network node coordinates. The settings for generating the random problem instances are given in Table 5.2.

Table 5.2 Settings for generating ROSSP instances

Factors	Setting
Number of jobs (n)	5, 10, 20
Number of machines (m)	5, 10, 20
Uniform distribution of operation processing time (PT)	(1, 10), (1, 100)
Uniform distribution of network node coordinates (NC)	(1, 10), (1, 100)

Let PT and NC denote the variation of the uniform distribution for operation processing time and the uniform distribution for the network node coordinates respectively, and let L and H denote the low variation and high variation for PT and NC respectively. There are 36 combinations of settings in total. One instance is generated for each setting and the computational results are shown in Table 5.3. In Table 5.3, column 1 is the index of the problem instance. The number of jobs and number of machines for each instance are shown in columns 2 and 3 respectively. The settings of PT and NC are provided in columns 4 and 5 respectively. The lower bound of the ROSSP instance makespan is computed based on the lower bound computational method described in subsection 5.6.2 and the TSP tour length is obtained by solving the corresponding TSP problem optimally. Therefore, the lower bound is a strong lower bound for the ROSSP, meaning that it is possible for the problems' optimal makespan to be equal to the lower bound value. The initial solutions are constructed using the method described in subsection 5.6.1 and the corresponding

makespans are given in column 7 in Table 5.3. The TS solutions' makespans are listed in column 9. The relative deviation from the lower bound of the initial solution in percentage and the TS solution is given in columns 8 and 10 respectively. The computation time (excluding the lower bound computation time) of the TS algorithm is given in column 11.

Table 5.3 Computational results

problem	*n*	*m*	*PT*	*NC*	Lower bound	Initial solution	Initial solution Dev. (%)	TS solution	TS solution Dev. (%)	TS computation Time
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)
1	5	5	*L*	*L*	50	57	14.00	50	0.00	0.84
2	5	5	*L*	*H*	345	390	13.04	348	0.87	5.91
3	5	5	*H*	*L*	346	354	2.31	346	0.00	0.75
4	5	5	*H*	*H*	648	720	11.11	648	0.00	0.06
5	5	10	*L*	*L*	80	81	1.25	80	0.00	0.95
6	5	10	*L*	*H*	303	356	17.49	324	6.93	2.86
7	5	10	*H*	*L*	636	636	0.00	636	0.00	0.39
8	5	10	*H*	*H*	747	845	13.12	747	0.00	1.11
9	5	20	*L*	*L*	144	144	0.00	144	0.00	0.91
10	5	20	*L*	*H*	268	313	16.79	268	0.00	4.34
11	5	20	*H*	*L*	1285	1285	0.00	1285	0.00	0.06
12	5	20	*H*	*H*	1417	1417	0.00	1417	0.00	0.61
13	10	5	*L*	*L*	96	106	10.42	98	2.08	1.58
14	10	5	*L*	*H*	434	511	17.74	441	1.61	12.69
15	10	5	*H*	*L*	651	658	1.08	651	0.00	1.75
16	10	5	*H*	*H*	922	1083	17.46	933	1.19	31.30
17	10	10	*L*	*L*	95	111	16.84	104	9.47	5.00
18	10	10	*L*	*H*	362	453	25.14	410	13.26	38.22
19	10	10	*H*	*L*	644	722	12.11	650	0.93	103.51
20	10	10	*H*	*H*	881	1062	20.54	914	3.75	158.48
21	10	20	*L*	*L*	137	148	8.03	137	0.00	12.44
22	10	20	*L*	*H*	427	525	22.95	514	20.37	94.32
23	10	20	*H*	*L*	1293	1293	0.00	1293	0.00	5.30
24	10	20	*H*	*H*	1374	1475	7.35	1375	0.07	291.13
25	20	5	*L*	*L*	154	168	9.09	162	5.19	6.61
26	20	5	*L*	*H*	475	632	33.05	541	13.89	59.03
27	20	5	*H*	*L*	1243	1249	0.48	1248	0.40	14.54
28	20	5	*H*	*H*	1639	1763	7.57	1663	1.46	50.96
29	20	10	*L*	*L*	169	182	7.69	173	2.37	16.92
30	20	10	*L*	*H*	476	630	32.35	584	22.69	201.29
31	20	10	*H*	*L*	1325	1343	1.36	1339	1.06	73.46
32	20	10	*H*	*H*	1667	1838	10.26	1751	5.04	395.40
33	20	20	*L*	*L*	172	227	31.98	220	27.91	24.11
34	20	20	*L*	*H*	556	740	33.09	719	29.32	251.36
35	20	20	*H*	*L*	1300	1369	5.31	1300	0.00	407.34
36	20	20	*H*	*H*	1682	2209	31.33	1917	13.97	384.55
Average							**12.57**		**5.11**	**73.89**

Comparing columns 8 and 10, we find that the tabu search algorithm is very efficient in reducing the makespans of the initial solutions. In average, the relative deviation from the lower bound makespan is reduced from 12.57% to 5.11%. In order to illustrate the effect of the four factors on the relative deviation, an analysis of variance (ANOVA) was conducted and the results are shown in Table 5.4.

Table 5.4 ANOVA for TS solution relative deviations

Source	Sum of Squares	D.F.	Mean Square	*F*-Ratio	*P*-Value
N	564.974	2	282.487	8.05	0.0017
M	178.001	2	89.0003	2.54	0.0965
NC	200.742	1	200.742	5.72	0.0234
PT	455.751	1	455.751	12.99	0.0012
RESIDUAL	1017.25	29	35.0775		
TOTAL (CORRECTED)	2416.72	35			

Table 5.5 ANOVA for TS computation time

Source	Sum of Squares	D.F.	Mean Square	*F*-Ratio	*P*-Value
n	147347	2	73673.5	9.81	0.0006
m	70923.1	2	35461.6	4.72	0.0168
NC	47463	1	47463	6.32	0.0177
PT	38764.4	1	38764.4	5.16	0.0307
RESIDUAL	217757.0	29	7508.85		
TOTAL (CORRECTED)	522254.0	35			

From Table 5.4 we get *P*-value(*n*) = 0.0017 and *P*-value(*PT*) = 0.001, which are considerably smaller than $\alpha = 0.01$. Therefore, we have strong evidence to conclude that the number of jobs (*n*) and the distribution of operation processing times (*PT*) affect the final relative deviation obtained by the tabu search algorithms.

Similarly, ANOVA was conducted for the computation time and the results are summarized in Table 5.5. It shows that *P*-value(*n*) = 0.0006, which is significantly smaller than $\alpha = 0.01$. Hence, we have strong evidence to conclude that the

computation time of the TS algorithm for the ROSSP is influenced by the number of jobs.

5.7 Conclusions

In this chapter, we developed two tabu search algorithms to minimize the makespan for the OSSP and the ROSSP respectively. New neighborhoods were defined for both of the two problems. Moreover, an exact method was developed to remove infeasible move of operations quickly. To overcome the disadvantage of the existing back jump tracking technique, we modified the existing back jump tracking technique by generating different starting solutions and storing them in the elite solution list before a search procedure is launched. Our tabu search algorithms were tested based on both benchmark and randomly generated problem instances. The computational results show that our algorithm performs very well for both small sized with less than 100 operations as well as large sized problems with up to 400 operations. Moreover, our algorithms were able to find the optimal solutions for many of the OSSP and ROSSP problem instances. For those problem instances whose optimal solutions were not verified, the gaps between the makespans obtained and the optimal solution makespans or lower bound of makespans were quite small.

Chapter 6 Conclusions and Future Research

In this dissertation, we have covered a number of routing shop scheduling problems, i.e. the single machine scheduling problem with unequal release dates, the single machine scheduling problem with setup times, the open shop scheduling problem, and the routing open shop scheduling problem. In this chapter, we summarize the research work conducted in this study and provide some concluding remarks to close our research dissertation. We also provide suggestions for future research at the end of this chapter.

6.1 Summary and Conclusions

In the last four decades, researchers in the area of operations research have worked on different types of manufacturing problems, and numerous exact and heuristics have been proposed to solve these problems. However, most of the research work conducted in the literature ignores product traveling times, machine traveling times and machine setup times. To improve the overall manufacturing system performance, we propose both exact and heuristics to solve the routing shop scheduling problems that consider machine or product transportation or setup times.

Chapter 3 of this dissertation provides a detailed description of a branch-and-bound algorithm that was developed to minimize the total weighted tardiness for the single machine scheduling problem with unequal release dates. To make the branch-and-bound algorithm more efficient, three global dominance rules, one local dominance rule, and a lower bound computational method were introduced to prune the search tree. The performance of the branch-and-bound algorithms was tested based on randomly generated problem instances and the computational results show that the

branch-and-bound algorithm is efficient in solving the SMSP with unequal release dates.

To solve large sized single machine scheduling problems, a brand new heuristic, named overlapped neighborhood search (ONS) algorithm, is proposed in Chapter 4. The ONS algorithm is a general-purpose algorithm that is applicable to those problems whose solutions can be represented by permutations. The basic idea of the ONS algorithm is to divide the permutation of a solution into overlapping blocks; each block is then explored independently. The whole solution can be further improved due to the existence of the overlaps between adjacent blocks. Results from the computational experiments conducted in this research work show that the ONS algorithm is efficient in solving both the single machine total weighted tardiness problem with unequal release dates and the single machine total weighted tardiness problem with sequence dependent setup times.

In Chapter 5, we introduce new neighborhoods for both the open shop scheduling problem and the routing open shop scheduling problem. In order to remove the infeasible move of operations quickly, we propose an exact feasibility checking method for the OSSP and the ROSSP. The computational results for the OSSP show that our tabu search algorithms performed very well for both small sized problems with less than 100 operations and large sized problems with up to 400 operations. Moreover, our tabu search algorithms were able to find the optimal solutions for most of the problem instances. For the ROSSP, computational experiments were conducted based on randomly generated problem instances. A lower bound computational method, which is tighter than the existing lower bound, is developed for ROSSP. The computational results show that the tabu search algorithm embedded with new

neighborhoods and new back jump tracking strategy is able to improve the initial solutions significantly within a short computation time.

The main contributions of this study are summarized below.

(1) New global dominance rules and a lower bound computational method were developed and integrated into a branch-and-bound algorithm. The computational experiments, which are based on randomly generated problem instances, show that the global dominance rules and the lower bound computational method proposed in this study are efficient in reducing the size of the search tree;

(2) A brand new general purpose heuristic, called overlapped neighborhood search (ONS) algorithm, was developed for machine scheduling problems whose solutions can be represented with permutations, such as various single machine scheduling problems, the traveling salesman problem (TSP), linear ordering problems (LOP), quadratic assignment problems (QAP) and bandwidth reduction problems (BRP) etc. Our computational results show that the ONS algorithm is an efficient and fast local search algorithm for solving single machine scheduling problems;

(3) New neighborhood structures were defined for two multi-machine scheduling problems, the open shop scheduling problem, and the routing open shop scheduling problem. Numerical experiments show that our new neighborhoods, exact feasibility checking methods, and new back jump tracking strategy, which are embedded into the tabu search algorithms, are efficient in minimizing the makespan for the OSSP and ROSSP.

6.2 Future Research

In Chapter 3, we demonstrated that the branch-and-bound algorithm that was developed for the single machine total weighted tardiness problem is an efficient algorithm. It is expected that the performance of the branch-and-bound algorithm can be improved further by incorporating sophisticated search strategies. Moreover, a stronger and more efficient lower bound computational method is also helpful for improving the performance of the exact algorithm.

The ONS algorithm proposed in Chapter 4 is a promising general-purpose algorithm for sequencing problems, such as the traveling salesman problem with time windows, quadratic assignment problems, linear ordering problems, and bandwidth reduction problems, etc. Comprehensive computational experiments are required in order to test the performance of the ONS algorithm for different types of sequencing problems. Furthermore, we provided several block improvement procedures for the ONS algorithm in Chapter 4. The efficiency of different BIPs should be evaluated to provide guidelines for other researchers. Another promising area is to extend the ONS algorithm to accept "worse-of" solutions based on a probabilistic strategy to jump out of the local optima. In addition, the ONS algorithm can be hybridized with other meta-heuristics, such as tabu search, simulated annealing, ant colony optimization algorithm, etc. to improve its performance.

As the exact feasibility checking method proposed in Chapter 5 is also applicable to the classical job shop and the open shop scheduling problems, it is possible to improve the performance of the existing algorithms developed in the literature for these two problems further by applying the exact feasibility checking method. Further work is expected to evaluate the influence of the exact feasibility checking method on the existing job shop and open shop scheduling algorithms.

References

[1] Abdul-Razaq, T.S., Potts, C.N. and Van Wassenhove, L.N. A survey of algorithms for the single machine total weighted tardiness scheduling problem, Discrete Applied Mathematics, 26, pp.235–253. 1990.

[2] Adams, J., Balas, E. and Zawack, D. The shifting bottleneck procedure for job shop schedule, Management Science, 34(3), pp.391-401. 1988.

[3] Agin, N. Optimum seeking with branch and bound, Management Science, 13, pp.176-185. 1966.

[4] Akturk, M.S. and Ozdemir, D. An exact approach to minimizing total weighted tardiness with release dates, IIE Transactions, 32, pp.1091-1101. 2000.

[5] Akturk, M.S. and Ozdemir, D. A new dominance rule to minimize total weighted tardiness with unequal release times, European Journal of Operational Research, 135, pp. 394-412. 2001.

[6] Applegate, D. and Cook, W. A computational study of the job-shop scheduling problem, ORSA Journal on Computing, 3, pp.149-156. 1991.

[7] Armentano, V.A. and de Araujo, O.C.B. Grasp with memory-based mechanism for minimizing total tardiness in single machine scheduling with setup times, Journal of Heuristics, 12, pp.427-446. 2006.

[8] Asano, M. and Ohta, H. A heuristic for job shop scheduling to minimize total weighted tardiness, Computers and Industrial Engineering, 42, pp.137-147. 2002.

[9] Averbakh, I. and Berman, O. Routing two-machine flowshop problems on networks with special structure, Transportation Science, 30(4), pp.303–314. 1996.

[10] Averbakh, I. and Berman, O. A simple heuristic for m-machine flow-shop and its applications in routing-scheduling problems, Operations Research, 47(1), pp.165–170. 1999.

[11] Averbakh, I., Berman, O. and Chernykh, I.D. A 6/5 –approximation algorithm for the two-machine routing open-shop problem on a 2-node network, European Journal of Operational Research, 166 (1), pp.3–24. 2005.

[12] Averbakh, I., Berman, O. and Chernykh, I.D. The routing open-shop problem on a network: complexity and approximation, European Journal of Operational Research, 173, pp.531-539. 2006.

[13] Aydin, M.E. and Fogarty, T.C. A simulated annealing algorithm for multi-agent systems: a job shop scheduling application, Journal of Intelligent Manufacturing, 15(6), pp. 805-814. 2004.

[14] Baker, K.R. and Bertrand, J.W.M. A dynamic priority rule for scheduling against due-dates, Journal of Operations Management, 3, pp.37-42. 1982.

[15] Baker, K.R. Introduction to sequencing and scheduling. John Wiley & Sons, Inc. New York. 1974.

[16] Balas, E. and Vazacopoulos, A. Guided local search with shifting bottleneck for job-shop scheduling, Management Science, 44(2), pp.262-275. 1998.

[17] Balas, E. Machine sequencing via disjunctive graphs: An implicit enumeration algorithm, Operations Research, 17, pp.941-957. 1969,

[18] Balas, E. On the facial structure of scheduling polyhedra, Mathematical Programming Study, 24, pp.179-218. 1985.

[19] Baptiste, P., Carlier, J. and Jouglet, A. A branch-and-bound procedure to minimize total tardiness on one machine with arbitrary release dates, European Journal of Operational Research, 158, pp.595-608. 2004.

[20] Baptiste, P., Le Pape, C. and Nuijten, W. Constraint-Based Scheduling, Applying Constraint Programming to Scheduling Problems, International Series in Operations Research and Management Science, 39, Kluwer. 2001.

[21] Barnes, J.W. and Chambers, J.B. Solving the job shop scheduling problem using tabu search, IIE Transactions, 27, pp.257-263. 1994.

[22] Belouadah, H., Posner, M.E. and Potts, C.N. Scheduling with release dates on a single machine to minimize total weighted completion time, Discrete Applied Mathematics, 36, pp.213-231. 1992.

[23] Berry, W. L., Penlesky, R. J. and Vollmann, T. E. Critical ratio scheduling dynamic due date procedures under demand uncertainty, IIE Transactions, 16 (1), pp. 81-89. 1984.

[24] Bierwirth, C. A generalized permutation approach to job shop scheduling with genetic algorithms, OR Spectrum, 17, pp.87-92. 1995.

[25] Blackstone, J., Phillips, D. and Hogg, G. A state-of-the-art survey of dispatching rules for manufacturing job shop operations, International Journal of Production Research, 20(1), pp.27-45. 1982.

[26] Błażewicz, J., Domschke, W. and Pesch, E. The job shop scheduling problem: Conventional and new solution techniques, European Journal of Operational Research, 93, pp.1-33. 1996.

[27] Błażewicz, J., Dror, M. and Weglarz, J. Mathematical programming formulations for machine scheduling: A survey, European Journal of Operational Research, 51, pp.283-300. 1991.

[28] Błażewicz, J., Ecker, K.H., Pesch, E., Schmidt, G. and Węglarz, J. Scheduling Computer and Manufacturing Processes. (2nd ed.) Springer, New York. 2001.

[29] Boffey, T.B. A note on minimal Hamilton path and circuit algorithms, Operational Research Quarterly, 24(3), pp.437-439. 1973.

[30] Bratley, P., Florian, M. and Robillard, P. On Sequencing with Earliest Starts and Due-Dates with Application to Computing Bounds for the (n/m/ G/Fmax) problem, Naval Research Logistics Quarterly, 20, pp.57-67. 1973,

[31] Brucker, P., Hurink, J., Jurish, B. and Wostmann, B. A branch and bound algorithm for the open-shop problem, Discrete Applied Mathematics, 76, pp. 43-59. 1997.

[32] Brucker, P., Jurisch, B. and Sievers, B. A branch and bound algorithm for the job-shop scheduling problem, Discrete Applied Mathematics, 49, pp.107-127. 1994.

[33] Campbell, H.G., Dudek, R.A. and Smith, M.L. A heuristic algorithm for the n job, m machine sequencing problem, Management Science, 16B, pp.630-637. 1970.

[34] Campos, V., Laguna, M. and Martí, R. Context-independent scatter and tabu search for permutation problems, INFOMRS Journal on Computing, 17(10) pp.111-122. 2005.

[35] Caridi, M. and Cavalieri, S. Multi-agent systems in production planning and control: an overview, Production Planning and Control, 15, pp. 106–118. 2004.

[36] Carlier, J. and Pinson, E. An algorithm for solving the job-shop problem, Management Science, 35, pp.164-176. 1989.

[37] Caseau, Y. and Laburthe, F. Disjunctive scheduling with task intervals. Working paper, Ecole Normale Supérieure, Paris. 1995.

[38] Chou, F.D., Chang, T.Y. and Lee, C.E. A heuristic algorithm to minimize total weighted tardiness on a single machine with release times, International Transactions in Operational Research, 12, pp.215-233. 2005.

[39] Christofides, N. Graph Theory: An algorithmic approach. Academic Press, New York. 1975.

[40] Christofides, N. The shortest Hamiltonian Chain of a graph, SIAM Journal on Computing, 19(4), pp.689-696. 1970.

[41] Chu C. A branch-and-bound algorithm to minimize total tardiness with different release dates, Naval Research Logistics, 39, pp.265-283, 1992.

[42] Chu, C. and Portmann, M.C. Some new efficient methods to solve the $n|1|r_i|\sum T_i$ scheduling problem, European Journal of Operational Research, 58, pp.404-413. 1992.

[43] Cochrane, E.M. and Beasley, J.E. The co-adaptive neural network approach to the Euclidean traveling salesman problem, Neural Networks, 16, pp.1499-1525. 2003.

[44] Colorni, A., Dorigo, M.and Maniezzo, V. Distributed Optimization by ant colonies. In Varela, F. and Bourgine, P. Editors. Proceedings of the European Conference on Artificial Life. Elsevier, Amsterdam. 1991.

[45] Croes, G.A. A method for solving traveling salesman problem, Operations Research, 6, pp.791-812. 1958.

[46] Dauzere-Peres, S. and Lasserre, J.B. A Modified Shifting Bottleneck Procedure for Job-Shop Scheduling, International Journal of Production Research, 31, pp.923-932. 1993.

[47] Dell'Amico, M. Shop problems with two machines and time lags, Operations Research, 44, pp.777–787. 1996.

[48] Dell'Amico, M. and Trubian, M. Applying tabu-search to the job shop scheduling problem, Annals of Operations Research, 41, pp.231-252. 1993.

[49] Dorigo, M. and Gambardella, L.M. Ant colonies for the traveling salesman problem, Biosystems, 43, pp.73-81. 1997.

[50] Dorigo, M., Maniezzo, V. and Colorni, A. The ant system: optimization by a colony of cooperating agents, IEEE Transactions on Systems, Man and Cycbernetics, Part B (26), pp.29-41. 1996.

[51] Dorndorf, U. and Pesch, E. Evolution based learning in a job shop scheduling environment, Computers & Operations Research, 22, pp.25-40. 1995.

[52] Dorndorf, U., Pesch, E. and Phan, H.T. Solving the open shop scheduling problem, Journal of Scheduling, 4, pp.157-174. 2001.

[53] Du, J. and Leung, J.Y.T. Minimizing total tardiness on one machine is NP-hard, Mathematics of Operations Research, 15, pp.483-495. 1990.

[54] Elmaghraby, S.E. The one-machine sequencing problem with delay costs, Journal of Industrial Engineering, 19, pp.105-108. 1968.

[55] Emmons, H. One-machine sequencing to minimize certain functions of job tardiness, Operations Research, 1, pp.701-715. 1969.

[56] Feo, T. and Resende, M.G.C. Greedy randomized adaptive search procedures, Journal of Global Optimization, 6, pp.109-133. 1995.

[57] Fernandes, E.R. and Ribeiro, C.C. A multistart constructive heuristic for sequencing by hybridization using adaptive memory, Electronic Notes in Discrete Mathematics, 19, pp.41-47. 2005.

[58] Fiala, T. An algorithm for the open-shop problem, Mathematics of Operations Research, 8(1), pp.100-109. 1983.

[59] Framinan, J. M., Gupta, J. N. D. and Leisten, R. A review and classification of heuristics for permutation flow-shop scheduling with makespan objective, Journal of the Operational Research Society, 55(12), pp. 1243-1255. 2004.

[60] França, P.M, Mendes, A. and Moscato, P. A Memetic algorithm for the total tardiness single machine scheduling problem, European Journal of Operational Research, 132, pp.224-242. 2001.

[61] Fry, T.D., Vickens, L., MacLeod, K. and Fernandez, S. A heuristic solution procedure to minimize t bar on a single machine, Journal of the Operational Research Society, 40, pp.293-297. 1989.

[62] Gagné, C., Price, W.L. and Gravel, M. Comparing an ACO algorithm with other heuristics for the single machine problem with sequence dependent setup times, The Journal of the Operational Research Society, 53, pp.895-906. 2002.

[63] Garey, M.R. and Johnson, D.S. Computers and intractability: A guide to the theory of NP-completeness. San Francisco: W.H. Freeman. 1979.

[64] Ghedira, K. and Ennigrou, M. How to Schedule a Job Shop Problem through Agent Cooperation Source, Lecture Notes in Computer Science, 1904, pp.132 – 141. 2000.

[65] Giffler, B. and Thompson, G.L. Algorithms for solving production scheduling problems, Operations Research, 8, pp.487-503. 1960.

[66] Glover, F. Tabu search – Part I, ORSA Journal on Computing, 1, pp.190-206. 1989.

[67] Glover, F. Tabu search – Part II, ORSA Journal on Computing, 2, pp.4-32. 1990.

[68] Glover, F. Tabu search and adaptive memory programming - advances, applications and challenges. In: Barr RS, Helgason RV, Kennington JL. (Eds.), Computing tools for modeling, optimization and simulation: interfaces in computer science and operations research, Kluwer, Boston, pp.1–75. 1996.

[69] Glover, F., Taillard, E. and de Werra, D. A user's guide to tabu search, Annals of Operations Research, 41, pp.3-28. 1993.

[70] Goldberg, A.V., Grigoriadis, M.D. and Tarjan, R.E. Efficiency of the Network Simplex Algorithm for the Maximum Flow Problem, Working paper, Report No. STAN-G-89-1248. Department of Computer Science, Stanford University. 1989.

[71] Gonçalves, J. F., José de, J. Mendes, M. and Resende, M.G.C. A hybrid genetic algorithm for the job shop scheduling problem, European Journal of Operational Research, 167, pp.77-95. 2005.

[72] Gonzalez, T. and Sahni, S. Open shop scheduling to minimizing finish time, Journal of ACM, 23, pp. 665-679. 1976.

[73] Graham, R.L., Lawler, E.L., Lenstra, J.K. and Rinnooy Kan AHG. Optimization and approximation in deterministic sequencing and scheduling: a survey, Annals of Discrete Mathematics, 5, pp.287-326. 1979.

[74] Guerét, G. and Prins, C. Classical and new heuristics for the open-shop problem: a computational evaluation, European Journal of Operational Research, 107, pp.306-314. 1998.

[75] Gupta, J.N.D. A functional heuristic algorithm for the flow-shop scheduling problem, Operations Research Quarterly, 22, pp.39-47. 1971.

[76] Gupta, S.R. and Smith, J.S. Algorithms for single machine total tardiness scheduling with sequence dependent setups, European Journal of Operational Research, 175, pp.722-739. 2006.

[77] Hasija, S. and Rajendran, C. Scheduling in flowshops to minimize total tardiness of jobs, International Journal of Production Research, 42(11), pp. 2289-2301. 2004.

[78] Haupt, R. A survey of priority-rule based scheduling, OR Spectrum, 11, pp.3-16. 1989.

[79] Ho, J.C. and Chang. Y.L. A new heuristic for the n-job, M-machine flow shop problem, European Journal of Operational Research, 52, pp.194-202. 1991.

[80] Holsenback, J.E. and Russell, R.M. A heuristic algorithm for sequencing on one machine to minimize total tardiness, Journal of the Operational Research Society, 43, pp. 53-62. 1992.

[81] Huang, W. Q. and Wang, L. A local search method for permutation flow shop scheduling, Journal of the Operational Research Society, 57(10), pp.1248-1251. 2006.

[82] ILOG. ILOG CPLEX, Reference and User's Manual, version 10.0, ILOG. 2006.

[83] Johnson, S.M. Optimal two- and three-stage production schedules with set up times included, Naval Research Logistics Quarterly, 1, pp.61-68. 1954.

[84] Jones, A. and Rabelo, L.C. Survey of job shop scheduling techniques, NISTIR––National Institute of Standards and Technology, Gaithersburg, MD. 1998.

[85] Jouglet, A., Baptiste, P. and Carlier, J. Branch-and-bound algorithms for total weighted tardiness. In Leung J. Y-T. (Ed.) Handbook of Scheduling: Algorithms, Models, and Performance Analysis. Chapter 13, Chapman & Hall. CRC Press, 2004.

[86] Jouglet, A., Savourey, D., Carlier, J. and Baptiste, P. Dominance-based heuristics for one-machine total cost scheduling problem, European Journal of Operational Research, 184, pp.879-899. 2008.

[87] Kahn, A.B.N. Topological sorting of large networks, Communications of the ACM, 5, pp.558-561. 1962,

[88] Kirkpatrick, S., Gelatt, C.C. and Vecchi, M.P. Optimization by simulated annealing, Science, 220, pp.671-680. 1983.

[89] Kis, T. and Pesch, E. A review of exact solution methods for the non-preemptive multiprocessor flowshop problem, European Journal of Operational Research, 164(3), pp.592-608. 2005.

[90] Kyparisis, G. J. and Koulamas, C. Flexible flow shop scheduling with uniform parallel machines, European Journal of Operational Research, 168(3), pp.985-997. 2006.

[91] Lee, C.Y. and Chen Z.L. Machine scheduling with transportation considerations, Journal of Scheduling, 4(1), pp.3-24. 2001.

[92] Lee, C.-Y. and Strusevich, V.A. Two-machine shop scheduling with an uncapacitated interstage transporter, IIE Transactions, 37, pp.725-736. 2005.

[93] Lee, Y.H., Bhaskaran, K. and Pinedo, M. A heuristic to minimize the total weighted tardiness with sequence-dependent setups, IIE Transactions, 29, pp.45-52. 1997.

[94] Lenstra, J.K, Rinnooy Kan A.H.G. and Brucker, P. Complexity of machine scheduling problems, Annals of Discrete Mathematics, 1, pp.343-362. 1977.

[95] Liaw, C.F. An iterative improvement approach for the nonpreemptive open shop scheduling problem, European Journal of Operational Research, 111, pp.509-517. 1998.

[96] Liaw, C.F. A tabu search algorithm for the open shop scheduling problem, Computers & Operations Research, 26, pp.109-126. 1999.

[97] Liaw, C.F. A hybrid genetic algorithm for the open shop scheduling problem, European Journal of Operational Research, 124, pp.28-42. 2000.

[98] Liaw, C.F. An efficient tabu search approach for the two-machine preemptive open shop scheduling problem, Computers & Operations Research, 30, pp.2081-2095. 2003.

[99] Liaw, C.F. Scheduling preemptive open shops to minimize total tardiness, European Journal of Operational Research, 162 (1), pp.173-183. 2005.

[100] Lin, S. and Kernighan, B.W. An effective heuristic algorithm for the traveling salesman problem, Operations Research, 21, pp.498-516. 1973.

[101] MacChiaroli, R. and Riemma, S. A negotiation scheme for autonomous agents in job shop scheduling, International Journal of Computer Integrated Manufacturing, 15, pp.222-232. 2002.

[102] Maggu, P.L. and Das, G. On the 2 × n sequencing problem with transportation time of jobs, Pure and Applied Mathematical Sciences, 12, pp.1–6. 1980.

[103] Manne, A.S. On the job shop scheduling problem, Operations Research, 8, pp.219-223. 1960.

[104] Martin, P. and Shmoys, D.B. A New Approach to Computing Optimal Schedules for the Job-Shop Scheduling Problem, Proceedings of the 5th International Conference on Integer Programming and Combinatorial Optimization, IPCO'96. 1996.

[105] Mason, S.J., Kutanoglu, E. and Fowler, J.W. Manufacturing and Logistics Applications of Multiple Orders Per Job Scheduling, Proceedings of 14th Industrial Engineering Research Conference, IIE, Atlanta, GA. 2005.

[106] Matsuo, H., Suh, C.J. and Sullivan, R.S. A controlled search simulated annealing method for the general job shop scheduling problem. Working paper 03-04-88, University of Texas Austin. 1988.

[107] Mattfeld, D. C. Evolutionary search and the job shop: investigations on genetic algorithms for production scheduling. Physica-Verlag, Heidelberg. 1996.

[108] Minton, S., Johnston, M.D., Philips, A.B. and Laird, P. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems, Artificial Intelligence, 58, pp.161-205, 1992.

[109] Montagne, E.R. Sequencing with time delay costs. Industrial Engineering Research Bulletin, Arizona State University, Tucson. 1969.

[110] Nakano, R. and and Yamada, T. Conventional genetic algorithm for job shop problems, in: R.K. Belew and L.B. Booker (eds.), Proc. 4th. International Conference on Genetic Algorithms, Morgan Kaufmann, pp.474-479. 1991.

[111] Nowicki, E., Smutnicki, C. A fast taboo search algorithm for the job shop problem, Management Science, 42(6), pp. 797-813. 1996.

[112] Nuijten, W., Le Pape, C. Constraint-based job shop scheduling with ILOG SCHEDULER, Journal of Heuristics, 3, pp.271-286. 1998.

[113] Onwubolu, G.C. and Mutingi, M. Genetic algorithm for minimizing tardiness in flow-shop scheduling, Production Planning & Control, 10(5), pp.462-471. 1999.

[114] Or, I. Traveling salesman-type combinatorial problems and their relation to the logistics of blood banking. Ph.D. Thesis, Department of Industrial Engineering and Management Sciences, Northwestern University. 1976.

[115] Osman, I.H. and Potts, C.N. Simulated Annealing for permutation flow-shop scheduling, Omega, 17, pp. 551-557. 1989.

[116] Pan, J.C.H., Chen, J.S. and Chao, C.M. Minimizing tardiness in a two-machine flow-shop, Computers and Operations Research, 29(7), pp. 869-885. 2002.

[117] Panwalker, S. and Iskander, W. A survey of scheduling rules, Operations Research, 25(1), pp.45-61. 1977.

[118] Pinedo, M. and Singer, M. A shifting bottleneck heuristic for minimizing the total weighted tardiness in job shop, Naval Research Logistics, 46, pp.1-12. 1999.

[119] Pinedo, M. Scheduling: theory, algorithms, and systems. 2nd ed., Prentice Hall, New Jersey. 2002.

[120] Potts, C.N, and van Wassenhove, L.N. A branch and bound algorithm for the total weighted tardiness problem, Operations Research, 33, pp.363-377. 1985.

[121] Potts, C.N. and van Wassenhove LN. Single machine tardiness sequencing heuristics, IIE Transactions, 23, pp.346-354. 1991.

[122] Puente, J., Diez, H.R., Varela, R., Vela, C.R and Hidalgo, L.P. Heuristic rules and genetic algorithms for open shop scheduling problem, Current Topics in Artificial Intelligence Lecture Notes in Computer Science, 3040, pp.394-403. 2004.

[123] Rachamadugu, R.M.V. and Morton, T.E. Myopic heuristics for the single machine weighted tardiness problem. Working Paper #28-81-82, Graduate School of Industrial Administration, Carnegie-Mellon University. 1981.

[124] Ragatz, G.L. A branch-and-bound method for minimum tardiness sequencing on a single processor with sequence dependent setup times. In: Proceedings: twenty-fourth annual meeting of the Decision Sciences Institute, pp.1375-1377. 1993.

[125] Raman, N, Rachamadugu, R.V. and Talbot, F.B. Real-time scheduling on an automated machine center, European Journal of Operational Research, 40, pp.222-242. 1989.

[126] Ramudhin, A. and Marier, P. The generalized shifting bottleneck procedure, European Journal of Operational Research, 93, pp.34-48. 1996.

[127] Rayward-Smith, V.J. and Rebaine, D. Open shop scheduling with delays, Theoretical Informatics and Applications, 26, pp.439-448. 1992.

[128] Reeves, C. A genetic algorithm for flow shop sequencing, Computers & Operations Research, 22, pp.5-13. 1995.

[129] Resende, M.G.C. and Ribeiro C.CGreedy randomized adaptive search procedures. In Handbook of Metaheuristics. Glover, F.W, G.A. Kochenberger, eds. International Series in Operations Research and Management Science. Kluwer Academic Publishers: Boston, pp.219-250. 2003.

[130] Rinnooy Kan, A.H.G., Lageweg, B.J., and Lenstra, J.K. Minimizing total costs in one-machine scheduling, Operations Research, 23, pp. 908-927. 1975.

[131] Roy, B. and Sussmann, B. Les problemes d'ordonnancement avec contraintes disjonctives, SEMA, Note D.S. No. 9. 1964.

[132] Rubin, P.A. and Ragatz, G.L. Scheduling in a sequence dependent setup environment with genetic search, Computers and Operations Research, 22, pp.85-99. 1995.

[133] Sen, T., Sulek, J.M. and Dileepan, P. Static scheduling research to minimize weighted and unweighted tardiness: A state-of-the-art survey, International Journal of Production Economics, 83, pp.1-12. 2003.

[134] Senthilkumar, P. and Shahabudeen, P. GA based heuristic for the open job shop scheduling problem, International Journal of Advanced Manufacturing Technology, 30, pp.297–301. 2006.

[135] Shapiro, J. A survey of Lagrangian techniques for discrete optimization, Annals of Discrete Mathematics, 5, pp.113-138. 1979.

[136] Singer, M. and Pinedo, M. A computational study of branch and bound techniques for minimizing the total weighted tardiness in job shops, IIE Transactions Scheduling and Logistics, 30, pp. 109-118. 1998.

[137] Somhom, S., Modares, A. and Enkawa, T. A self-organising model for the traveling salesman problem, The Journal of the Operational Research Society, 48, pp.919-928. 1997.

[138] Sourd, F. and Nuijten, W. Multiple-machine lower bounds for shop-scheduling problems, INFORMS Journal on Computing, 12, pp.341-352. 2000.

[139] Steiglitz, K. and Weiner, P. Some improved algorithms for computer solution of the traveling salesman problem. Proceedings of the 6th Annual Allerton Conference on Communication, Control and Computing. Department of Electrical Engineering and the Coordinated Science Laboratory, University of Illinois, Urbana, IL. 814-821. 1968.

[140] Strusevich, V.A. A heuristic for two-machine open shop scheduling problem with transportation times, Discrete Applied Mathematics, 93, pp.287-304. 1999.

[141] Sun, D., Batta, R. and and Lin, L. Effective job shop scheduling through active chain manipulation, Computers & Operations Research, 22, pp.159-172. 1995.

[142] Taillard, E. Some efficient heuristic methods for the flow shop sequencing problem, European Journal of Operational Research, 47, pp.65-74. 1990.

[143] Taillard, E. Benchmarks for basic scheduling problems, European Journal of Operational Research, 64, pp.278-285.1993.

[144] Taillard, E. Parallel tabu search technique for the job shop scheduling problem, ORSA Journal on Computing, 6, pp.108-117. 1994.

[145] Taillard, E. An introduction to ant systems. In Computing Tools for Modeling, Optimization and Simulation. Laguna, M., J.L. González-Velarde, eds. Kluwer, Boston. pp.131-144. 2000.

[146] Tamura, M., Taga, A., Kitagawa, S. and Banbara, M. Compiling Finite Linear CSP into SAT, Lecture Notes in Computer Science, 4204, pp.590-603. 2006.

[147] Tan, K.C. and Narasimhan R. Minimizing tardiness on a single processor with sequence dependent setup times: a simulated annealing approach, Omega, 25(6), pp.619-634. 1997.

[148] Tan, K.C., Narasimhan, R., Rubin, P.A. and Ragatz, G.L. A comparison of four methods for minimizing total tardiness on a single processor with sequence dependent setup time, Omega, 28, pp.313-326. 2000.

[149] Vaessens, R.J.P. Aarts, E.H.L. and Lenstra, J.K. Job shop scheduling by local search, Journal on computing, 8, pp.302-317. 1996.

[150] van Laarhoven, P.J.M., Aarts, E.H.L. and Lenstra, J.K. Job shop scheduling by simulated annealing, Operations Research, 40, pp.113-125. 1992.

[151] Vepsalainen, A.P.J. and Morton, T.E. Priority rules for job shops with weighted tardiness costs, Management Science, 33, pp.1035-1047. 1987.

[152] Widmer, M. and Hertz, A. A new heuristic method for the flow shop sequencing problem, European Journal of Operational Research, 41, pp.186-193. 1989.

[153] Wilkerson, L. J. and Irwin, J. D. An Improved Method for Scheduling Independent Tasks, IIE Transactions, 3(3), pp.239 - 245. 1971.

[154] Zeng, L., Ong, H.L. and Ng, K.M. A generalized crossing local search method for solving vehicle routing problem, The Journal of the Operational Research Society, 58, pp.528-532. 2007.

Appendix A

Proof of global dominance rules

Global dominance rule 1A: Let J_i and J_k be two jobs ($i, k \in S$). If

(a) $r_i \leq r_k$,

(b) $w_i \geq w_k$,

(c) $p_i = p_k$, and

(d) $d_i \leq \max\left\{d_k, \max\left\{r_k, LBC_{B_k}\right\} + p_k\right\}$,

then J_i precedes J_k.

Proof:

Consider two jobs J_i and J_k in a schedule which satisfy the above conditions but with J_k preceding J_i. We assume that the initial start time of J_k and the completion time of J_i are S_k and C_i respectively. Suppose the positions of the two jobs are interchanged. Note that this interchange is valid as conditions (a) and (c) are also satisfied, and the completion time of the other jobs does not change.

Figure A1 Initial schedule

From Figure A1 we can obtain

$$S_k + p_i + p_k \leq C_i. \qquad \text{(A1)}$$

It is noted that (A1) is applicable to all the three global dominance rules.

The net decrease in tardiness of J_i by interchanging J_i and J_k is

$$w_i \max\{C_i - d_i, 0\} - w_i \max\{S_k + p_i - d_i, 0\}. \quad \text{(A2)}$$

Similarly, the net increase in tardiness of J_k by interchanging J_i and J_k is

$$w_k \max\{C_i - d_k, 0\} - w_k \max\{S_k + p_k - d_k, 0\}. \quad \text{(A3)}$$

Case 1. $d_i \le d_k$. We consider the following three sub-cases:

(1) If $d_i \le d_k < C_i$, the net decrease in tardiness due to the interchange of the two jobs is

$$\begin{aligned}
&\text{(A2) } - \text{ (A3) } = [w_i \max\{C_i - d_i, 0\} - w_i \max\{S_k + p_i - d_i, 0\}] - \\
&\left[w_k \max\{C_i - d_k, 0\} - w_k \max\{S_k + p_k - d_k, 0\}\right] \\
&= w_i[C_i - \max\{S_k + p_i, d_i\}] - w_k[C_i - \max\{S_k + p_k, d_k\}] \\
&= C_i(w_i - w_k) + [w_k \max\{S_k + p_i, d_k\} - w_i \max\{S_k + p_i, d_i\}] \\
&\ge C_i(w_i - w_k) + [w_k \max\{S_k + p_i, d_i\} - w_i \max\{S_k + p_i, d_i\}] \\
&= [C_i - \max\{S_k + p_i, d_i\}](w_i - w_k) \ge 0.
\end{aligned}$$

(2) If $d_i \le C_i \le d_k$, the net decrease in tardiness due to the interchange of the two jobs is

$$\begin{aligned}
&\text{(A2)} - \text{(A3)} = [w_i \max\{C_i - d_i, 0\} - w_i \max\{S_k + p_i - d_i, 0\}] - \\
&[w_k \max\{C_i - d_k, 0\} - w_k \max\{S_k + p_k - d_k, 0\}] \\
&= w_i[C_i - d_i - \max\{S_k + p_i - d_i, 0\}] + w_k \max\{S_k + p_k - d_k, 0\} \\
&\ge w_i \min\{C_i - d_i, C_i - S_k - p_i\} + w_k \max\{S_k + p_k - d_k, 0\} \ge 0.
\end{aligned}$$

(3) If $C_i < d_i \le d_k$, the net decrease in tardiness due to the interchange of the two jobs is

$$\begin{aligned}
&\text{(A2)} - \text{(A3)} = [w_i \max\{C_i - d_i, 0\} - w_i \max\{S_k + p_i - d_i, 0\}] - \\
&[w_k \max\{C_i - d_k, 0\} - w_k \max\{S_k + p_k - d_k, 0\}] = 0.
\end{aligned}$$

Thus the net decrease in tardiness is nonnegative, and so the interchange of the two jobs can be made without increasing the total weighted tardiness.

Case 2. $d_i \le \max\{r_k, LBC_{B_k}\} + p_k$. We have $S_k \ge \max\{r_k, LBC_{B_k}\}$ because $\max\{r_k, LBC_{B_k}\}$ is the earliest start time of job J_k. Therefore, we obtain

$d_i \le S_k + p_k \le C_i - p_i$.

The net decrease in tardiness due to the interchange of the two jobs is

$$
\begin{aligned}
&(\mathrm{A}2) - (\mathrm{A}3) = [w_i \max\{C_i - d_i, 0\} - w_i \max\{S_k + p_i - d_i, 0\}] - \\
&[w_k \max\{C_i - d_k, 0\} - w_k \max\{S_k + p_k - d_k, 0\}] \\
&= [w_i(C_i - d_i) - w_i(S_k + p_i - d_i)] - [w_k \max\{C_i - d_k, 0\} - w_k \max\{S_k + p_k - d_k, 0\}] \\
&= w_i(C_i - S_k - p_i) - w_k[\max\{C_i - d_k, 0\} - \max\{S_k + p_k - d_k, 0\}] \\
&= w_i(C_i - S_k - p_i) - w_k[\max\{C_i, d_k\} - \max\{S_k + p_k, d_k\}] \\
&\ge w_i[C_i - \max\{C_i, d_k\} + \max\{S_k + p_k, d_k\} - (S_k + p_k)].
\end{aligned}
$$

We consider the following two sub-cases:

(1) If $C_i \ge d_k$, then the above expression can be simplified as follows:

$$
\begin{aligned}
&w_i[C_i - \max\{C_i, d_k\} + \max\{S_k + p_k, d_k\} - (S_k + p_k)] \\
&= w_i[\max\{S_k + p_k, d_k\} - (S_k + p_k)] \ge 0.
\end{aligned}
$$

(2) If $C_i < d_k$, then the above expression can be simplified as follows:

$$
\begin{aligned}
&w_i[C_i - \max\{C_i, d_k\} + \max\{S_k + p_k, d_k\} - S_k - p_k] \\
&= w_i[C_i - d_k + \max\{S_k + p_k, d_k\} - S_k - p_k] \\
&= w_i[\max\{S_k + p_k, d_k\} - d_k + (C_i - S_k - p_k)] > 0.
\end{aligned}
$$

Thus the net decrease in tardiness is nonnegative, and so the interchange of the two jobs can be made without increasing the total weighted tardiness.

Global dominance rule 2: Let J_i and J_k be two jobs ($i, k \in S$). If

(a) $r_i \le r_k$,

(b) $p_i = p_k$, and

(c) $d_k \ge UBC_S - Sum_{A_i}$,

then J_i precedes J_k.

Proof:

Consider two jobs J_i and J_k in a schedule which satisfy the above conditions but with J_k preceding J_i. We assume that the start time of J_k and the completion time of J_i are S_k and C_i respectively. Suppose the positions of the two jobs are interchanged. Note that this interchange is valid as conditions (a) and (b) are also satisfied, and the completion time of the other jobs does not change.

Based on the definition of UBC_S we know that $UBC_S - Sum_{A_i} \geq C_i$. From (c), we obtain $d_k \geq UBC_S - Sum_{A_i} \geq C_i > S_k + p_k$.

The net decrease in tardiness due to the interchange of the two jobs is

$$
\begin{aligned}
&(\text{A2}) - (\text{A3}) = [w_i \max\{C_i - d_i, 0\} - w_i \max\{S_k + p_i - d_i, 0\}] - \\
&[w_k \max\{C_i - d_k, 0\} - w_k \max\{S_k + p_k - d_k, 0\}] \\
&= w_i \max\{C_i - d_i, 0\} - w_i \max\{S_k + p_i - d_i, 0\} \\
&= w_i[\max\{C_i, d_i\} - \max\{S_k + p_i, d_i\}] \geq 0.
\end{aligned}
$$

Thus the net decrease in tardiness is nonnegative, and so the interchange of the two jobs can be made without increasing the total weighted tardiness.

Global dominance rule 3: For any job J_k ($k \in S$), if $d_k \geq UBC_S$, then J_k can be assigned last. In the situation that there are $m \geq 1$ jobs satisfying $d_k \geq UBC_S$, then the *m* jobs can be assigned in the last *m* positions in any sequence without sacrificing the optimality of the schedule.

Proof:

As $d_k \geq UBC_S$, the tardiness of job J_k is zero for any position where it is placed in an active schedule. Therefore, assigning J_k at the last position will not affect other jobs.