Single machine two-agent scheduling problems with tardiness objectives

by

Jiaji Li

A Thesis submitted to the Faculty of Graduate Studies of

The University of Manitoba

In partial fulfilment of the requirements of the degree of

MASTER OF SCIENCE

I.H. Asper School of Business

Department of Supply Chain Management

University of Manitoba

Winnipeg, Manitoba, Canada

Copyright © 2018 by Jiaji Li

Abstract

In the last decade, study on two-agent scheduling has attracted lots of attention from many researchers due to its challenging complexity level and broad application in different areas. In this thesis, three two-agent scheduling problems are considered in which scheduling criteria of both agents need to be satisfied simultaneously. The first problem is a single machine two-agent scheduling problem, with the objective to minimize the weighted number of tardy jobs from the first agent. Meanwhile, the upper bound of weighted number of tardy jobs from another agent could not be violated. The second and the third problems take order acceptance into account. In these two scheduling problems, some of the jobs could be rejected when the workload is limited. The objective of these two problems is to maximize the profits gained from the revenues of accepted jobs, while the scheduling criterion for the second agent (weighted number of tardy jobs) is bounded by a pre-determined value. A heuristic method and a meta-heuristic solution method based on Particle Swarm Optimization and Tabu Search are proposed to solve the problems. In addition, a dynamic programming based exact algorithm which could optimally solve the first problem is developed. Furthermore, a mathematical formulation of the third problem is also provided. The main objective of the thesis is to develop algorithms for solving three different two-agent scheduling problems as stated earlier.

Key words: two-agent, single machine, due date, number of tardy jobs, total tardiness

Acknowledgements

I would like to express my sincere gratitude to my advisor Dr. Yuvraj Gajpal and my co-advisor Dr. S. S. Appadoo for their support and encouragement during my study at the University of Manitoba. Their guidance has helped me overcome the difficulties in the early steps of my graduate study. I would also like to thank my committee members Dr. Sara Hajmohammad and Dr. Ruppa Thulasiram for their guidance and insightful comments on my thesis. At last, I want to express my gratitude to my parents. For their unparalleled care and love which has enabled me to pursue a master's degree in a foreign country.

Table of contents

Tables and figures

Chapter 1

Introduction

In the last decade study on two-agent scheduling problems attracted increasing attention from many researchers. There are mainly two reasons for the popularity of two-agent scheduling: (1) consideration of two agents in scheduling problems increases the computational complexity, enlarging the improvement space in solution approaches; (2) two-agent scheduling problems arising from real-life situations could be applied to many industries.

Multi-agent scheduling model introduces the concept of agent to the study of scheduling problems. In these problems, there are several jobs belonging to different agents and agents have their own criterion. When scheduling the jobs, different criterion should be satisfied. The rationality lies on the fact that manufacturers usually schedule jobs for different customers. Thus, it seems that the original setting of scheduling models with only one criterion is not applicable when manufacturers schedule jobs for different customers. Numbers of successful applications of two-agent scheduling problems in different industries also indicate the usefulness of the study on two-agent scheduling problems.

In the following subsections, general introductions to scheduling and its key elements are provided along with a subsection introducing the two-agent scheduling model and its applications.

1.1. Introduction to scheduling

Scheduling is a decision-making process dealing with problems of allocating resources to the tasks over given time periods and has been applied to many manufacturing and service industries (Pinedo, 2016). The aim of scheduling is to optimize certain objectives by utilizing the resources efficiently. Scheduling plays an important role in many manufacturing systems and information processing environments through facilitating the decision-making process.

Since the study on scheduling is applicable in many areas such as Management, Engineering and Manufacturing, tasks, resources and optimality criteria (objectives) may take different forms in different settings. For example, in manufacturing, scheduling is about determining the sequence of jobs to be processed on machines; however, in context of airports, scheduling might refer to coordinating the usage of runways at the airport among different airlines. Despite the differences, there is a standardized description of classical scheduling problems: allocating jobs to the machines, with the objective functions related to completion times (Fatos & Ajith, 2008).

1.1.1. Key elements in scheduling model

As mentioned above, there are three key elements that constitute a scheduling problem: machine environment, job characteristics and the optimality criterion. These three key elements constitute a scheduling problem. According to Graham (1979), scheduling problems are classified by their machine environments, job characteristics and optimality criterion, and is written in the form of a triplet: $\alpha \mid \beta \mid \gamma$.

1.1.2. Machine environments

In scheduling problems, machine environment refers to the machines on which the jobs are processed and appears in the first field of the triplet. Machine environment in different problems is mainly determined by the number of processors as well as the route which the jobs are processed by the machines.

Single machine scheduling model is a special case for scheduling model because only one machine is used to process the job. The machine environment of single machine is denoted by 1 in the α field. Though it is relatively simplified compared to other machine environments, researchers could better understand more complicated problem models by studying single machine problems. This is because there are some properties in single machine problems that do not exist in other scheduling problems. By means of these properties, researchers could propose heuristics for other problems (Pinedo, 2016). In addition, studying more complex problems such as multiple machine problems can be considered as special case of this problem, by reducing the number of machines to zero.

Parallel machine (P_m) problems could be regarded as an extension of the single machine problems. In this setting, there are *m* machines in parallel, and jobs could be completed on any one of the machines. Thus, decisions are made not only on the sequence of jobs to be processed, but also on the selection of machines to process the jobs (Mokotoff, 2001). Parallel machine problems assume that the machines in parallel process the jobs with the same speed. Thus, another scheduling problem considering parallel machines add the variant of speed at which the machines process the jobs. Scheduling problems with this machine environment is denoted as $Q_m| \beta | \gamma$.

There are some other machine environments dealing with multi-step operations. In flow shop problems (F_m), each job has to be processed on all the machines following a pre-determined route. Flexible flow shop problems generalize flow shop problems and parallel machine problems: jobs are completed after *c* steps in total, there might be several machines available at each step.

Job shop problems (J_m) and open shop problems (O_m) are the most complicated problem models in scheduling. In job shop problems, jobs might visit machines more than once. They become flexible job shop problems (FJ_c) when more than one machine is available at each step. The open shop problems eliminate the constraint of the pre-determined route for jobs to be processed and schedulers could determine the routes for every job.

1.1.3. Job characteristics

In the β field of the triplet notation, some specific characteristics of jobs with processing restrictions are specified. With these specific scheduling constraints, scheduling models could depict real-life situations more closely.

Following are some common job characteristics claimed in β field. Release date (r_j) is used when the jobs are not ready at the beginning. Preemption ($prmp$) means that interruption during the processing of jobs is allowed in scheduling, interrupted jobs could be processed afterwards. In cases where the jobs should be processed in given orders, i.e., a job could not be processed if specific jobs are not completed, precedence ($prec$) is claimed. Some job characteristics have an impact on the value of processing times. Learning effects (*LE*) in scheduling problems means that processing times of jobs decreases due to the improvement of proficiency on job processing.

In contrast, job deterioration specifies the conditions of which processing time of a job becomes larger when it is positioned closer to the end of the schedule.

1.1.4. Optimality criteria

After a schedule is proposed for a scheduling problem, it is evaluated by specific optimality criteria. Most of the objective functions/optimality criteria in scheduling are non-decreasing functions related to completion time. Objectives directly correlated to completion time (C) are: makespan (C_{max}), total completion time ($\sum C_i$), weighted sum of completion time ($\sum W_i C_i$). The makespan (C_{max}) denotes the highest values among all job completion times. Common objectives that are due-date related including total lateness ($\sum L_i, L_i = C_i - D_i$), tardiness ($\sum T_i, T_i = \max(C_i - D_i, 0)$). Note that the difference between tardiness and lateness is that tardiness is non-negative objective while lateness might be lower than zero. Another common due date-related objective is the number of tardy jobs ($\sum U_i$). Variable U_i is binary, taking the value of 1 when job i is tardy. When different weights are assigned to the jobs according to their priority level, the objective becomes weighted number of tardy jobs ($\sum W_i U_i$).

1.2. Two-agent scheduling

1.2.1. Introduction

Two-agent scheduling problem was first proposed by Baker & Smith, (2003) and Agnetis et al. (2004), in which jobs from two agents of different interests share common resources. Though in many two-agent scheduling models, the agents are in competition with each other, it may not always be the case. Two-agent scheduling problems could be classified into four categories

according to relationships between the job sets belonging to the agents, namely, Competing Agents, Interfering Sets, Multicriteria Optimization, Non-disjoint Sets (Agnetis et al., 2014).

Figure 1.1 Classifications of two-agent scheduling problems (Agnetis et al., 2014)

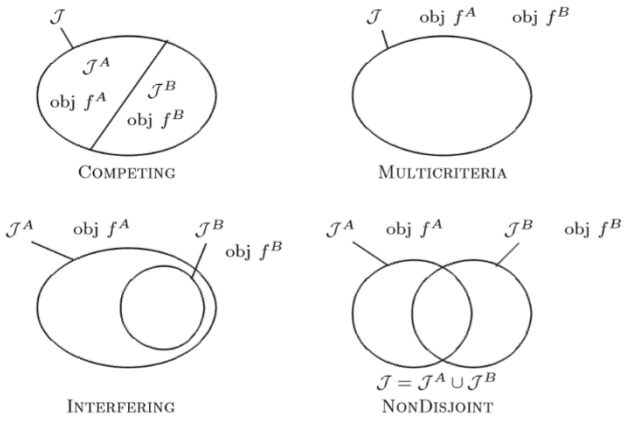

Figure 1.1 shows the relationships between the two job sets. As shown in the figure, there are two job sets: J^A, J^B, with complete job set J. The objectives of job set J^A and J^B are denoted by $obj\ f^A, obj\ f^B$ respectively. When jobs belonging to one of the job set do not belong to another, the problem is categorized as Competing Agents Scheduling problems. Multicriteria problems eliminate the job sets but retain objectives from different agents. Interfering problems refer to the problems in which the job set from one agent includes all the jobs while the job set belonging to other agent is its subset. Non-Disjoint problems are the ones in which the job sets share some common jobs.

After determining the two-agent problems, schedulers need to choose a solution approach to solve these problems. Widely used approaches include Feasibility Problem, Linear Combination of Criteria and Epsilon-Constraint Approach. The first approach tries to solve the problems by providing feasible solutions. Approach of Linear Combination of Criteria solves the problems by combining criterion for the agents into one objective function by assigning weights to them

($\alpha f^A + (1-\alpha) f^B$). The third approach, Epsilon-Constraint Approach often used in literature, solves the problem by optimizing the objective of agent *A*, meanwhile the criterion for agent *B* is bounded. The problems considered in this thesis follow the third approach for two-agent scheduling problem.

1.2.2. Applications

Studies on multi-agent scheduling involves many fields and industries. Brewer and Plott (1996) worked on a scheduling problem in context of railroad management. In their problem, access to tracks are sold to different companies and schedulers need to provide a feasible time schedule for the trains so that the trains do not interfere with each other. Schultz et al. (2002) focused on a scheduling problem in telecommunication industry. In their problem, different agents/enterprises share the usage of the satellite, and they have different preferences. For example, voice service may tolerate the loss of some data, but it is strict with delay requirements; in contrast, service of data transferring gives the top priority level to data loss. Leung et al. (2005) applied multi-agent scheduling problems to order scheduling problems where schedulers deal with orders sent from different customers with different requirements on processing resources and due dates. Pessan et al. (2008) worked on a rescheduling problem, in which some of the jobs are needed to be rescheduled due to the limitation of workload. These jobs are given higher priority level than other jobs. Study on multi-agent scheduling could also be applied to airports. Soomer and Franx (2008) worked on the application of multi-agent scheduling in airports, where number of runways for the take-offs and landing is limited. Different airlines compete on the usage of the runways, collaboration among airlines is also needed to exchange information.

1.3. Thesis overview and contributions

This thesis focuses on three two-agent scheduling problems: (1) single machine two-agent scheduling problem with the objective of minimizing weighted number of tardy jobs of the first agent, subject to the upper bound of weighted number of tardy jobs of the other agent; (2) single machine order acceptance two-agent scheduling problem with the objective of maximizing the benefits gained by total revenues from accepted jobs minus the penalty function of weighted lateness of the first agent, while maintaining the weighted number of tardy jobs from the second agent within a pre-determined level; (3) single machine order acceptance problem with the objective to maximize benefits gained by total revenues from accepted jobs minus the penalty function of weighted tardiness of the first agent, with an upper bound on weighted number of tardy jobs belonging to the second agent. We propose algorithms including two meta-heuristics (Particle Swarm Optimization, Tabu Search) and one heuristic to solve these problems. The contributions of this thesis are as follows. Firstly, the proposed algorithms (two meta-heuristics and one heuristic) effectively solved the three problems in this thesis. It is noteworthy to point out that some features are added to the meta-heuristics to obtain better solutions. In Particle Swarm Optimization, feature of parameter normalization effectively saves CPU time and improves the solution quality. A new method to update the parameters is also proposed in this thesis. The Particle Swarm Optimization proposed in Chapter 3 uses the parallel parameter mechanism, which has never been applied to this problem to the best of our knowledge. In Tabu Search, the feature of dynamic penalty factor also helps to improve the performance of the algorithm. Secondly, the studies in this thesis extend the literature on two-agent scheduling problem (1) by introducing a problem: “order acceptance two-agent scheduling problems with weighted tardiness penalty function”; (2) by introducing new meta-heuristics to solve the

problem of “two-agent scheduling problem with weighted number of tardy jobs’; (3) and by introducing a new meta-heuristic to solve the problem of “order acceptance two-agent scheduling problem with the penalty function of weighted lateness”.

The organization of the thesis is as follows. In Chapter 2, a two-agent scheduling problem with weighted number of tardy jobs is considered. In Chapter 3, order acceptance is taken into account, and two order acceptance two-agent scheduling problems are addressed; and the conclusion of this thesis is given in the last chapter.

Chapter 2

Two agent scheduling problem to minimize weighted number of tardy jobs

2.1. Introduction

Multi-agent scheduling models describe the conditions when schedulers deal with multiple job sets belonging to different agents or customers. Two-agent scheduling model is a special case of multi-agent scheduling models with only two agents. The study of two-agent scheduling is important for researchers as it provides the foundation for developing algorithms to solve multi-agent scheduling problems. In two-agent scheduling models, agents compete for the allocation of common resources. The objectives of the two agents conflict with each other. Therefore, organizations which process tasks from these agents need to strike a balance between the objectives of different agents. The two-agent scheduling problem is more challenging than single-agent scheduling problem because it has multiple objectives.

Two-agent scheduling has applications in artificial intelligence, decision theory, operations research, etc (Agnetis et al., 2004). Schultz et al. (2002) highlighted the application of two-agent scheduling problem in telecommunication to transfer data from different agents (e.g. voice service, file transfer). Perez-Gonzalez & Framinan (2014) discussed the application of two-agent scheduling model in machine maintenance between the maintenance department and the production department.

This chapter focuses on a two-agent scheduling problem with weighted number of tardy job objectives. This type of optimization problem on tardy jobs could be applied to various sector of the company. In service industry, decreasing the number of tardy jobs is important for improving customer satisfaction. In most industries, failure to complete jobs on time incur penalties. In supply chain management, lowering the number of tardy jobs is a sign of utilization of resources, and enhanced cooperation among different departments.

This chapter considers a single machine scheduling problem involving two agents. Each agent has a specific job set. These jobs are processed on a single machine and are released at time zero. Every job has its own processing time, due date and weight. The objective of this problem is to minimize the weighted number of tardy jobs of the first agent, subject to an upper bound of weighted number of tardy jobs of the second agent. According to the three-partition notation introduced by Graham (1979), the problem is denoted as: $1|\sum w_i^B U_i^B \leq Q|\sum w_i^A U_i^A$. This problem is NP-hard because it reduces to a single agent scheduling problem with weighted number of tardy job objectives when number of agents is set to one. The single machine scheduling problem with weighted number of tardy jobs objective has been proved to be NP-hard by Karp (1972). The exact algorithm developed in this chapter could not solve large scale optimization problem with decimal weights. Hence, this chapter proposes a Particle Swarm Optimization and Tabu Search algorithm to solve this problem.

Scheduling problem considered in this chapter could be applied many situations. For instance, for airports in Canada, they need to schedule the take-offs and landings of flights from different airlines. The airlines could be categorized into self-owned airline (Air Canada) and other airlines. The scheduling resources in this scenario are the runways at the airports. The flights are assigned with weights according to their levels of importance. The objective of airports is to minimize the

weighted number of tardy jobs from Air Canada, while the weighted number of tardy jobs from other airlines should be controlled within a pre-determined level. To the best of our knowledge, this is among the first study which studies meta-heuristics for two-agent single machine scheduling problem to minimize the weighted number of tardy jobs of first agent, subject to an upper bound of the weighted number of tardy jobs of another agent, i.e. $1|\sum w_i^B U_i^B \leq Q|\sum w_i^A U_i^A$. In this chapter we provide an exact algorithm and two meta-heuristics to solve this problem. We have also added problem features specific to these meta-heuristics.

The rest of the chapter is organized as follows. Section 2.2 reviews the literature related to this problem. Section 2.3 explains some notations related to this problem. In Section 2.4, we present the exact algorithm proposed in this chapterr. In Section 2.5, proposed PSO algorithm is presented. In Section 2.6, the proposed Tabu Search is presented. Section 2.7 reports the results of numerical experiment to evaluate the performances of proposed algorithms. In Section 2.8, a conclusion is provided to summarize this chapter.

2.2. Literature review

Moore (1968) is the pioneer in the study on tardy jobs. Moore (1968) addressed the complexity of the problem in which the objective is to minimize the number of tardy jobs on a single machine. They also proposed a fully-time polynomial algorithm. Karp (1972) proved that the minimization of weighted number of tardy jobs on a single machine is an NP-hard problem. M'Hallah and Bulfin (2003) proposed a branch and bound algorithm to solve this problem.

Two-agent scheduling models were first introduced by Baker & Smith (2003) and Agnetis et al. (2004). They worked on the problems with two agents, and two job sets belong to these agents respectively. The problems they worked are combinations of objectives in scheduling problems

including: makespan objective, completion time objective, maximal lateness objective, etc. These objectives are non-decreasing functions related to completion time (Mor & Mosheiov, 2010). Compared to bi-criteria or multi-objective scheduling problems, the inclusion of sets of jobs for individual agents makes the multi-agent scheduling models closer to real life situations. The major difference between the work of Baker & Smith (2003) and Agnetis et al. (2004) is the way objectives are optimized. Baker and Smith assigned weights to each objective and minimized the combined objective. Agnetis et al. (2004) used the idea of constrained optimization, in which the objective of the first agent is minimized while keeping the objective function of other agent within a pre-specified value. Agnetis et al. (2004) proved the complexity of different two-agent scheduling problems. According to the constrained optimization model proposed by Agnetis et al. (2004), the two-agent scheduling problem on single machine could be denoted as $1|\beta|f_A: f_B \leq Q$. The problem considered in this chapter follows the research direction of Agnetis et al. (2004). The growing number of research paper on two agent scheduling problem indicates the importance and challenges of two agent scheduling problem. (Sahu et al, (2017), Li et al. (2017), Wu et al. (2017), Wang et al. (2017), etc).

The remainder of the literature review focuses on two-agent scheduling problems with tardy job objectives. Cheng, Ng, & Yuan (2006) worked on a feasibility model of multi-agent scheduling ($1|\ |\sum_{1\leq j\leq n_j} w_j^i U_j^i \leq X_i$,(j= *1*,2,3...m)), where the objectives for all the agents are the weighted number of tardy jobs. The model tries to find a schedule which could satisfy all of the constraints ($w_j^i U_j^i \leq X_i$). They proved that the problem is NP-hard and provided a pseudo-polynomial algorithm with integral weights.

Ng, Cheng, & Yuan (2006) studied the two-agent scheduling problem to minimize the weighted completion time of first agent subject to an upper bound on the number of tardy jobs of another

agent. Ng, Cheng, & Yuan (2006) proved that the complexity of the problem and proposed a pseudo-polynomial algorithm. Cheng et al. (2011) extended the research of Ng, Cheng, & Yuan (2006) by considering the learning effect. They provided a branch and bound algorithm, and three simulated annealing algorithms to solve this problem. Wu (2013) studied the problem of minimizing total tardiness for the first agent subject to zero tardy jobs of the second agent. The author provided an exact algorithm as well as a genetic algorithm to solve this problem.

Cheng, Ng, & Yuan (2006) developed a dynamic programming based exact algorithm for a feasibility model. The dynamic programming algorithm of feasibility model is extended to solve the problem considered in this chapter. However, this exact algorithm has mainly two limitations. First limitation is that it could only solve problems with integral weights. Second limitation is that it could only solve relatively small instances. Hence, meta-heuristics are developed to solve larger problem instances. Meta-heuristics provide general framework and strategies for researchers to develop heuristics to solve their problems. The meta-heuristics conduct either single walks or multiple walks in the neighborhood area to search for better solutions. In this research, we apply both multiple-walk-method (PSO) as well as the single-walk method (Tabu Search) to solve the problem.

2.3. Problem description

The problem considers the jobs of two agents, namely agent *A* and agent *B*. Each agent has a set of jobs (J_A,J_B), containing n^A/n^B jobs respectively. Each job is assigned a processing time, a due date and a weight. All jobs are released at time 0, and processed on a single machine. Preemption of jobs is not allowed. The objective is to minimize the weighted number of tardy jobs of the first

agent subject to an upper bound on the weighted number of tardy jobs of another agent. Following notations are related to the problem description in this chapter.

n^A/n^B	number of jobs in agent *A*/ agent *B*
n	total number of jobs, $n = n^A + n^B$
l	index for A-jobs
M	index for B-jobs
j	index used to denote job set;
i	index used to denote jobs in job set *j*
J^A/J^B	job set belonging to agent A/ agent B, $J_A = \{J_1^A, J_2^A, J_3^A, \ldots J_{n^A}^A\}$, $J_B = \{J_1^B, J_2^B, J_3^B, \ldots J_{n^B}^B\}$
P_l^A/P_m^B	processing time of each job from agent *A*/agent *B* respectively
D_l^A/D_m^B	due date of each job from agent *A*/agent *B* respectively
W_l^A/W_m^B	weight of each job from agent *A*/agent *B* respectively
σ	ordered set of jobs already scheduled
$C_l^A(\sigma)/C_m^B(\sigma)$	completion time of each job from agent *A*/agent *B* respectively
U_i^j	index used to denote whether job is tardy, $\begin{cases} \text{if } C_i^j(\sigma) > D_i^j, U_i^j = 1 \\ \text{else } U_i^j = 0 \end{cases}$
Q	upper bound of wegithed number of tardy jobs of agent *B*

Given the notations above, problem can be represented as follows:

Objective function: Mini$mize \sum_{1\le i\le n^A} w_i^A U_i^A$ (2.1)

Subject to: $\sum_{1\le i\le n^B} w_i^B U_i^B \le Q$ (2.2)

Equation (2.1) shows the objective function of this problem, and equation (2.2) states that the upper bound *Q* of agent *B* could not be violated.

2.4. Proposed algorithms

In this section, an exact algorithm is proposed to obtain the optimal solution in small instances. Meta-heuristics including PSO and Tabu Search are proposed to solve the large instances.

2.4.1. Exact algorithm

The exact algorithm proposed in this chapter is developed from the feasibility model of multi-agent scheduling problem. In the feasibility model of multi-agent scheduling, there are *m* agents. Each of them has an upper bound of their objective. Cheng, Ng, & Yuan (2006a) considered a multi-agent scheduling problem, with weighted number of tardy job objectives ($1|| \sum_{1\le j\le n_j} w_j^i U_j^i \le Q_j$,(*j= 1*,2,3…*m*)). Here Q_j denotes the upper bound of weighted number of tardy jobs of agent *j*. The feasibility model determines whether a feasible solution can be found. The feasibility model does not minimize any objective function, while the problem considered in this chapter minimizes the objective function of agent A. The dynamic programming provided by Cheng, Ng, & Yuan (2006a) used for solving feasibility model can be extended to solve our problem. The dynamic programming of feasibility model uses the following property.

Property 1: *If the problem is feasible, then there is a feasible schedule in which early jobs are sequenced consecutively in the non-decreasing order of due dates (EDD order) at the beginning of the schedule.*

This property indicates that if an optimal partition of jobs in late job set and early set can be obtained, then the optimal solution can be obtained by arranging early set in *EDD* order. Based on this property, they developed a dynamic programming algorithm. The following equations show the initial condition and recursion function of the dynamic programming formulation for the feasibility model of two-agent scheduling problem, i.e. $1|\sum_{1\leq m\leq n^B} W_m^B \leq Q_2|\sum_{1\leq l\leq n^A} W_l^A \leq Q_1$.

Initial condition

$C(0, X_1, X_2) = 0$ if all $X_i \geq 0$;

$C(k, X_1, X_2) = +\infty$, if $k < 0$ or some $X_i < 0$ or $X_i > Q_i$;

Recursion function

$$C(\mathrm{k}, X_1, X_2) = \min \begin{cases} C(k\text{ - }1, X_1, X_2) + P_k + f(k, X_1, X_2) \\ C(k\text{ - }1, X_1 - W_k, X_2) \text{ or } C(k\text{ - }1, X_1, X_2 - W_k) \end{cases}$$

if $C(k\text{ - }1, X_1, X_2) + D_k \leq D_k$, $f(k, X_1\ X_2) = 0$

Otherwise $f(k, X_1\ X_2) = +\infty$

From the initial condition and recursion function above, it could be seen that this dynamic programming algorithm solves the feasibility problem by calculating the completing time of the last early job in the job set. When there are two agents in the problem ($1|\sum_{1\leq m\leq n^B} W_m^B \leq$

$Q_2|\sum_{1\le l\le n^A} W_l^A \le Q_1$), then this problem is feasible only if the value of the completion time $C(n, Q_1, Q_2)$ is finite.

Since we try to adopt the Epsilon-Constraint Approach to solve the two-agent scheduling problem: $1|\sum w_i^B U_i^B \le Q|\sum w_i^A U_i^A$, the dynamic programming algorithm could not be directly applied to solve our problem. Thus, some additional efforts are needed to develop an exact algorithm based on the dynamic programming algorithm. The following are the steps of the exact algorithm developed in this chapter, the idea of the exact algorithm is to obtain the optimal solution by giving different values to Q_1 and solve the feasibility model iteratively.

Step 1: Initialize

Step 1.1: Rearrange the jobs from agent *A* and agent *B* in *EDD* order.

Step 1.2: Set $X_1 = 1,\ X_2 = Q$.

Step 2: Use the dynamic programming to test feasibility of feasibility model $1|\ |\sum_{1\le l\le n_A} W_l^A \le Q_1 : \sum_{1\le m\le n_B} W_m^B \le Q_2$

If the feasibility model is infeasible,

$X_1 = X_1 + 1$, return to Step 2.

Else

Optimal solution *OPT* = X_1

Stop.

Generating the schedule

The steps shown above could only calculate the objective value of the optimal solution for this problem, it would be meaningless if the corresponding schedule could not be obtained. Therefore, an additional section to generate the actual schedule of the optimal solution is developed in the exact algorithm. The following pseudo code shows the steps of generating the schedule of the problem in which the optimal solution OPT has been obtained.

Step 1: Set $X_1 = OPT,\ X_2 = Q, k = n_1 + n_2$

Step 2: Do the following to find the early set jobs of the schedule.

If ($C(k, X_1, X_2)$= $C(k - 1, X_1, X_2) + P_k$)

Select job k into early set

Else

Select job into late set

{

If $i \in J_1, X_1 = X_1 - W_k$

Else $X_1 = X_1 - W_k$

}

Step 3: Stopping criterion

If k >0, return to step 2

Else, go to Step 4.

Step 4: Find the schedule by arranging jobs in set E in EDD order at the beginning of the schedule, and late jobs in any order of the sequence.

2.4.2. PSO algorithm

Particle Swarm Optimization (PSO) was first introduced by Kennedy and Eberhart (1995). It is inspired by the social behavior of fish and birds. This algorithm is a population-based algorithm, where individuals in the population are updated based on a mechanism obtained from fitness values of other members in the group. However, unlike other population-based algorithms, PSO

doesn't combine individuals directly, instead, each particle adjusts its velocity based on its own experience, as well as the best particle's experience (Pandey et al., 2010). It helps the individuals to move towards the best solution while keeping the stochastic nature of an individual particle. PSO has been successfully applied to scheduling problems in studies such as Yoshida et al. (1999), Tasgetiren et al. (2004), Liao et al. (2007), Sahu et al. (2017).

We use the following notations to describe PSO algorithm.

M Number of particle,

k Number of iterations.

x_{ij}^{k} Position value of i th particle of j th dimension in k th iteration,

X_{i}^{k} Position values of i th particle in k th iteration,

v_{ij}^{k} Velocity of i th particle of j th dimension in k th iteration.

V_{i}^{k} Velocity of i th particle in k th iteration.

c_1, c_2 Social cognitive numbers

In this PSO algorithm, total number of n dimensions are presented to correspond to the total number of jobs from the two agents. Jobs in scheduling are presented in the proposed PSO algorithm by means of velocity (v_{ij}^{k}) and position value (x_{ij}^{k}). The jobs are sequenced based on the values of these variables and the property shown before. The steps of PSO in this research are explained as follows:

Step 1: Initialize the parameters of velocity and position value for each particle.

Step 1.1: Set k=0, m= 50.

Step 1.2: Generate initial position value vector X_i^0 and velocity vector V_i^0 randomly.

Step 1.3: Use the Sequence Generator to generate actual schedule from position values.

Step 1.4: Initialize the position value vector and velocity vector of the personal best solutions and the global best solution.

Step 2: Do the following for each particle:

Step 2.1: k=k+1

Step 2.2: Update inertia weight as: $w^k = w^{k-1} * \alpha$, where α is decrement counter.

Step 2.3: Update velocity and position value using the formulas given below:

$$\mathrm{v}_{ij}^{k} = w^{k-1}\mathrm{v}_{ij}^{k} + c_1 * r_1 * z_1 + c_2 * r_2 * z_2;$$

$$x_{ij}^{k} = x_{ij}^{k} + \mathrm{v}_{ij}^{k};$$

Step 2.4: Normalize the velocity and position values between [-4,4].

Step 2.5: Use the Sequence Generator to generate actual schedule.

Step 2.6: Apply following local search schemes:

- Local search 1: Insertion local search
- Local search 2: Removal local search
- Local search 3: Swap local search

Step 2.7: Update the position values and velocities of personal best and global best.

Step 3: Stopping criterion.

If the number of iterations exceeds the maximum number of iterations, then stop; otherwise, go back to step 2.

Initialization of parameters

The initialization of this proposed PSO algorithm starts with assigning values of velocities and position values to all the particles randomly within the range of [-4, 4]. The following formulas are used to initialize the velocities and position values:

$$x_{ij}^{0} = X_{min} + (X_{max} - X_{min}) * U(0,1)$$

$$v_{ij}^{0} = V_{min} + (V_{max} - V_{min}) * U(0,1),$$

Where $X_{min} = -4,\ X_{max} = 4,\ V_{min} = -4, V_{max} = 4$, and $U(0,1)$ is a uniform random number between 0 and 1. Parameters c_1 and c_2 equal to 0.8, r_1 and r_2 are random numbers generated randomly within the range of [0, 1]. The parameter values are set by performing sensitivity analysis within limited CPU time.

Sequence Generator or mapping PSO parameter values with scheduling problem

The Sequence Generator could be understood as the tool to map the PSO parameters to the scheduling problem: it interprets the values of parameters into the actual schedule. The position value in different dimension of a particle is mapped as a sequencing of jobs in either early or late schedule of the job. The dimension represents the job. The positive position value represents the sequencing of jobs on early set while the negative position value represents the sequencing of jobs in late set. According to the Property 1, an optimal solution can be obtained if an optimal partition of early and late set is obtained. Therefore, the sequence generator is based on the

partition of jobs in early set *E* and late set *L*. It is ensured that all of jobs in early set *E* are completed before their due dates, and the weighted number of tardy jobs for agent *B* is within an upper bound Q. The following two-phase method is used to generate sequence from position value.

Step 1: Assigning jobs of agent *B* to early set *E* and late set *L*.

A job from agent *B* is selected for assigning either early set *E* or late set *L*. If position value of a job is positive, then it is placed into set *E*, otherwise, it is placed into set *L*. Jobs are selected for assignment in *EDD* order. During the assignment process if the assignment of a job in early set makes the job late then it is placed into late set *L*. Let σ^C represent the current partial sequence after assigning all the jobs from agent *B*, $\sigma^C = \{\sigma_E^C \cup \sigma_L^C\}$.

Step 2: Obtain feasible partial sequence

The sequence generated at the end of the step 1 might be infeasible in terms of upper bound *Q*. The sequence is made feasible by moving jobs from late set *L* into early set *E*. The jobs to be moved from late set *L* to early set *E* are selected on the basis of their position values. The job with highest position value is given first priority and the job with the lowest position value is given the last priority.

Step 3: Assigning jobs of agent *A* to set *E* and set *L*

A job from agent *A* is selected for its assignment in existing early set *E* or late set *L*. Unlike assignment of jobs of agent *B*, the jobs of agent *A* is selected on the basis of their position values.

Updating parameters

In the existing updating method, the value of z_1 is corresponded by the differences between the position value of personal best solution and position value of current solution. Similarly, the value of z_2 is corresponded by the differences between the position value of global best solution and position value of current solution. The proposed updating method is different from the existing updating method on setting the value of z_1 and z_2. We use following formulas to update velocity, position value and intertia.

$$v_{ij}^{k} = w^{k-1} v_{ij}^{k-1} + c_1 * r_1 * z_1 + c_2 * r_2 * z_2;$$

$$x_{ij}^{k} = x_{ij}^{k-1} + v_{ij}^{k};$$

Parameters of z_1 and z_2 are related to personal best and global best solution. The parameter z_1 equals to 1 when job *i* is in the set *E* in personal best solutions; otherwise, it takes the value of -1. Similarly, parameter z_2 takes the value of 1 when job *i* is in set *E* in global best solutions and equals to -1 when it is not. Parameters of c_1 and c_2 are social coefficient quantifying the extent the individual relies on its own experiment (personal best) or global best. Inertia *w* is the resistance of any physical object to any change in the state of motion, it controls velocity and direction of all the particles. We set inertia weight equaling to 0.9 in this algorithm.

Local Search

Local search is applied to improve the solution on every particle at the end of each iteration. There are three kinds of local search applied in this chapter. These local search schemes are all applied to improve the solution generated by position values. It should be noted that the local search strategies consider feasible solutions only. Situations like violated due dates for jobs on set *E* or violation of the upper bound *Q* are not considered. Let σ represent the sequence of jobs.

In sequence σ, jobs in set *E* are sequenced consecutively in *EDD* order, and jobs in set *L* are placed in any order after the last job of *E*. The following local search schemes are repeatedly used till improvement in the solution is encountered.

- Local search 1: Insertion local search

Insertion local search tries to improve the solution by inserting jobs belonging to agent *A* from late set *L* into early set *E*. This local search scheme is also called "backfilling strategy" in Computer Science literature. An insertion is feasible if the insertion of a job does not violate the due dates of other jobs in *E* under σ. All the jobs of agent *A* in *L* are considered for possible insertion into schedule σ_E. The job which provides the best solution is selected for insertion.

- Local search 2: Removal local search

Removal local search is used to move jobs of agent *B* from early set *E* to late set *L*. From all feasible removals only one removal is selected randomly, because removal of job from early set *E* to late set *L* does not improve the objective function. However, the removal operator helps to improve the solution in future by bringing the jobs of agent *A* from late set *L* to early set *E*.

- Local search 3: Swap local search

Swap local search tries to improve the results by swapping the jobs in early set *E* and late set *L*. Evaluate all the feasible swaps which does not violate the due dates in *E* and upper bound of weighted number of tardy jobs of agent *B*. From all the feasible swaps, select the swap leading to the best solution.

Normalization of parameters

In the existing PSO, the position values and velocities are not restricted. However, we keep position value and velocity within some range. This process is called normalization. The normalization of parameters is one the new features introduced in this chapter. Values of velocity and position become very large after some iterations. Thus, it is difficult to remove a job from early set to late set after few times of iterations and vice versa. In this chapter, we normalize the velocity values and position values by regulating them within the range of [-4,4]. Parameters exceeding the upper limit take the value of 4. Similarly, parameters that are lower than the lower limit take the value of -4 respectively.

2.4.3. Tabu Search

Tabu search algorithm is proposed by Glover (1989) and is applied to solve various problems. Tabu Search focuses on combinatorial problems by providing near-optimal solutions. It is the process of searching better results in the neighborhood area to minimize the objective function. The idea of Tabu Search is to escape from local optimal solution by using flexible structure, conditions which prohibit or free the exploitation in the search area, and memory function mechanism such as defining the span of memory.

Tabu Search improves the solutions by exploring different kinds of moves. In this chapter, moves are defined as swap move, insertion move and removal move. Tabu Search uses three terminologies: *Tabu list, Tabu tenure, Aspiration rule*. The terminology *Tabu List* is created to store the moves which are prohibited in the next few steps. The terminology *Tabu Tenure* is used to define the duration of *Tabu List*. The terminology *Aspiration Level* is used to state the conditions in which the moves in *Tabu list* could be disregarded. In contrast with PSO proposed before, Tabu search also explores the infeasible area by considering solutions of which the

weighted number of tardy jobs of agent *B* exceeds its upper bound *Q*. We use following objective function to calculate the fitness value of a solution, $f(s) = f_1(s) + (f_2(s) - Q) * \gamma$, where $f(s)$ denotes the fitness value of the solution *s*, $f_1(s)$ denotes the weighted number of tardy jobs of agent *A*, $f_2(s)$ denotes the weighted number of tardy jobs of agent *B*, γ denotes the penalty factor.

In our Tabu Search algorithm, *Tabu List* is defined as a list of moves which are prohibited lasting for the number of *t* (i.e. *Tabu Tenure*) iterations. Following terminologies can be used to describe the steps of Tabu Search.

k Number of iterations.

s Current solution

N(s,k) Neighborhood of solution *s* in iteration *k*.

*s** The best solution found so far.

The pseudo code of Tabu Search is described below.

Step 1: Initialization

Step 1.1: Set *k*=0, *t* =100, γ=500.

Step 1.2 Generate an initial solution *s*, *s**=*s*.

Step 2: Do the following till the termination criteria is not met

Step 2.1: Generate all neighborhood solution *N(s,k).*

Step 2.2: Choose the *s'* with the best *f(s')* in *N(s,k).*

Step 2.3: Update dynamic penalty factor γ.

Step 2.4: Update s^*.

Step 2.5: $k=k+1$.

Step 3: Stopping criterion

If ($k>1000$), stop.

Otherwise, go to step 2.

Generate initial solution

To generate an initial solution, first all jobs are arranged in *EDD* order. The jobs are inserted one by one in the early set *E*. If placing a job into early set leads to the tardiness of this job or tardiness of existing early job, then it is placed in the late set *L*. The initial solution of Tabu Search is obtained by first scheduling jobs from set *E* in *EDD* order and then schedule the jobs from set *L* in any order of sequence.

Dynamic penalty factor

In this chapter, we used the dynamic penalty factor γ which was used by Cordue et al. (2001) for solving vehicle routing problem. After an iteration, if the objective function of agent B is feasible, then set $\gamma=\gamma/2$, otherwise, set $\gamma=\gamma*2$. When current solution is feasible the penalty factor is lowered to allow the exploration of infeasible solution; when current solution is infeasible, the penalty factor is increased to move the solution towards feasible region.

2.5. Numerical results

In this section, numerical experiment and analysis are conducted to evaluate the performances of the proposed algorithms. The effects of added feature on PSO algorithm and Tabu Search are also analyzed.

2.5.1. Instance generation

In this chapter, we generate 40 instances for this problem in total. The number of jobs of each agent ranges from 5 to 500. The processing time of jobs are generated randomly within the range of [1, 25], while weights are generated randomly within the range of [1, 5]. Due dates are generated by means of the formula: $d_i = Min + (Max - Min) * \mathrm{U}(0,1)$. Here *Min* equals to the value of 10 percent of sum of processing times, and *Max* equals to the value of 70 percent of sum of processing times. The value of Q is generated by the following formula: $Q = f_{B(\min)} + \alpha f_{B(max)}$ as suggested by Agnetis et al. (2009), where $f_{B(\min)}$ denotes the lowest value of the weighted number of tardy jobs of agent *B*. It is obtained by sequencing all the jobs from agent *B* in *EDD* order. The term $f_{B(max)}$ denotes the maximal value of weighted number of tardy jobs of agent *B*, it is obtained by sequencing all the agent *B* jobs after the agent *A* jobs. The term α is a random number ranging from 0.2 to 0.5.

2.5.2. Numerical analysis

The following notations are used for evaluating the performance of the algorithms.

OPT: Optimal objective function value generated by the exact algorithm.

ABS: Absolute objective function value generated by an algorithm.

APD: Absolute percentage of deviation of an algorithm from the optimal value.

RPD: Relative percentage of deviation of an algorithm from the best value.

CPU: Running time of the proposed algorithm.

The exact algorithm, the proposed PSO algorithm and the Tabu Search algorithm are coded in C++ programming language, the numerical experiments are implemented in a computer environment with AMD Opteron 2.3GHz with 256 GB RAM on Windows 7. It is worthwhile to mention that the exact algorithm of dynamic programming requires $n_1 \times n_2 \times Q$ memory allocations. Due to the restriction of C++ programming memory, the exact algorithm could solve instances containing jobs up to 280 jobs. Within the capacity of the exact algorithm, we evaluate the performance of algorithms based on the absolute values and its *APD*. When evaluating the performance of algorithms in large instances, where the exact algorithm could not solve, *RPD* and *CPU time* are used.

2.5.2.1 Numerical experiment for small problem instances

Since the exact algorithm has limited capacity on solving the problem, it is mainly used to evaluate the accuracy of other two algorithms within its capacity. The performances of proposed PSO and Tabu Search are evaluated by comparing the solutions with the optimal solution *OPT* obtained from the exact algorithm. Here we use absolute percentage deviation (APD) to evaluate the performances of proposed PSO and Tabu Search. It is obtained from the following equation: $APD = (ABS - OPT)/OPT * 100\%$.

Table 2.1 presents the numerical result of proposed algorithms for small problem instances. The proposed PSO algorithm provides the results that differ from the optimal solution by 4.99% on

average. The APD for proposed PSO ranges from 3.16% to 100%. The average APD for proposed Tabu Search algorithm is only 0.24%. The proposed PSO could find optimal solutions for 26 instances out of 32 instances, while the Tabu Search could find optimal solutions for almost every instance except one instance, which are considered to be a very good result for meta-heuristics.

Table 2.1 Numerical results in small instances

Number	Q	Exact algorithm	PSO		Tabu Search	
		OPT	ABS	APD	ABS	APD
5	4	1	1	0	1	0
6	4	1	1	0	1	0
7	9	3	3	0	3	0
8	8	7	7	0	7	0
9	6	6	6	0	6	0
10	4	4	4	0	4	0
11	3	21	21	0	21	0
12	13	2	2	0	2	0
13	13	1	1	0	1	0
14	11	2	2	0	2	0
15	10	7	7	0	7	0
16	17	5	5	0	5	0
17	4	12	12	0	12	0
18	6	9	9	0	9	0
19	6	8	8	0	8	0
20	14	3	3	0	3	0
21	11	8	8	0	8	0
22	10	7	7	0	7	0
23	15	10	10	0	10	0
24	11	17	17	0	17	0
30	12	8	8	0	8	0
40	15	12	12	0	12	0
50	36	1	2	100	1	0
60	21	26	26	0	26	0
70	38	13	14	7.7	13	0
80	29	29	30	3.45	29	0
90	32	22	22	0	22	0
100	40	23	24	4.35	23	0

110	88	8	10	25	8	0
120	41	32	33	3.16	32	0
130	92	12	13	8.33	12	0
140	88	13	14	7.69	14	7.69
Average		10.41	10.77	4.99	10.43	0.24

2.5.2.2 Numerical experiment for small problem instances

Table 2.2 reports the numerical result of PSO algorithm and Tabu Search algorithm for large problem instances. The performances in large instances are evaluated by comparing the relative performance of the two meta-heuristics. The RPD value is used here to evaluate the relative performances of proposed PSO and Tabu Search in large instances. The RPD is calculated as : $RPD = (ABS - ABS_{best})/ABS_{best} * 100\%$. The values of RPD for PSO and Tabu Search show that PSO is away from the Tabu Search by 11.77%. In terms of CPU time, Tabu Search also outperforms PSO. The average CPU time for Tabu Search is 1213.74 seconds, while the average CPU time of PSO is 13897.07 seconds. The average value of RPD shows that the proposed Tabu Search performs better than proposed PSO algorithm.

Table 2.2 Numerical experiment for large problem instances

Number	Q	PSO			Tabu		
		ABS	RPD	CPU	ABS	RPD	CPU
150	95	16	6.67	245.11	15	0	83.53
200	133	13	8.33	951.63	12	0	198.37
250	58	93	12.05	2929.86	83	0	374.08
300	80	118	3.51	6484.32	114	0	672.9
350	192	62	5.08	10080.4	59	0	1110.45
400	205	93	17.72	22193.3	79	0	1629.04
450	255	95	20.25	30605.9	79	0	2359.95
500	292	94	20.51	37686	78	0	3281.63
Average		73	11.77	13897.07	64.875	0	1213.74

2.5.2.3 Effects of normalization in PSO

We test the effects of normalization in PSO by comparing the performance of PSO algorithm with PSO algorithm without normalization. The results are reported in Table 2.3. The values of objective function of PSO without normalization is away from the optimal solution by 21.68%, and it equals to 4.99% for proposed PSO algorithm. The gap in terms of performance between the PSO algorithm with normalization and the PSO without normalization becomes larger in large problem instances. The PSO algorithm with normalization is 12.52 % away from the best results obtained from the meta-heuristics, while PSO without normalization is away from it by 100.77%.

Table 2.3 Effects of normalization in PSO algorithm

	Small problem instances		Large problem instances	
	Average ABS	Average APD	Average ABS	Average RPD
PSO without normalization	13.19	21.68	130.25	100.77
PSO with normalization	10.69	4.99	73	11.77

Based on the results in Table 2.3, it could be seen that the normalization effectively improves the results. The normalization is one of the new feature of the proposed PSO which could be applied to improve the performance of PSO in future research.

2.5.2.4 Effects of dynamic penalty factor in Tabu Search

Dynamic penalty factor γ is used to calculate the fitness value of solutions in Tabu Search. It is used to explore the search towards feasible and infeasible regions. Effects of dynamic penalty factor is tested by comparing the performances of Tabu search with dynamic penalty and that of Tabu Search with constant penalty.

Table 2.4 presents the numerical result of Tabu Search with dynamic factor and Tabu Search with constant factor. In small instances, the values of objective function of the Tabu Search with without dynamic penalty is 0.86% away from the optimal solution, and the proposed Tabu Search is 0.24% away from optimal solution. In large instances, the average RPD for the proposed Tabu Search is 0, and the values of objective function of the Tabu Search with dynamic factor is 23.17% . The results clearly indicate that changing γ dynamically helps to improve the performance of Tabu Search.

Table 2.4 Effects of penalty factor on Tabu Search

	Small problem instances		Large problem instances	
	Average ABS	Average APD	Average ABS	Average RPD
Tabu Search (with dynamic factor)	10.53	0.86	64.88	23.17
Tabu Search (without dynamic factor)	10.44	0.24	53.5	0

2.6. Conclusion

In this chapter, we focused on a two-agent scheduling problem with weighted number of tardy job objectives. We proposed an exact algorithm and introduce PSO and Tabu Search to solve this problem. The performance of these algorithms was evaluated by two types of numerical experiments performed on randomly generated problem instances. Because the capacity of the exact algorithm is limited to 280 jobs, the performance of these algorithms are evaluated in small problem instances and large problem instances. In small problem instances, the performances of the two meta-heuristics are compared with the optimal solutions obtained by the exact algorithm. In large problem instances, we only compare the solutions of the two meta-heuristics proposed in this chapter.

The proposed PSO and Tabu Search provides good solutions in small instances, with average percentage deviation of 4.99% and 0.24% respectively. In large instances, Tabu Search provides better solutions and could save 91% CPU time compared with PSO. The PSO algorithm could be further improved by allowing infeasible solutions in search space.

We also contribute to this problem by adding some features to the proposed meta-heuristics. In PSO, we propose a normalization method after updating the parameters in each iteration. In Tabu Search, we add a dynamic penalty factor to evaluate fitness values of moves. Both these features improve the performance of these meta-heuristics.

Chapter 3

Two-agent scheduling with order acceptance objectives

3.1. Introduction

Order acceptance has been studied by researchers for the last few decades. In many cases, the number of orders accepted by a firm is not necessarily positively related to its profit, especially when the capacity is limited. Firms need to reject some of the orders to maximize their profit when the associated costs for some of the orders exceed their revenues. Accepting orders without consideration might directly lead to the increase of workload. When the workload is excessively heavy compared to the capacity of the firm, it might incur late deliveries of orders, therefore decreasing the level of customer satisfaction or even losing the important customers. Therefore, study focusing on order acceptance has attracted the attention of many researchers.

In the field of scheduling, researchers brought order acceptance into consideration in the last few decades. They study scheduling problems in which decision makers need to decide which jobs to be processed and consider the sequence of the accepted jobs. There is a trade-off between the revenues and penalty function while deciding acceptance or rejection of jobs. The penalty functions are always some functions related to due dates, such as lateness, tardiness, etc. The objective of order acceptance scheduling problem is to maximize the profit gained by the revenues of accepted orders, minus the specific penalty function.

Some researchers tried to apply the study on order acceptance problems to multi-agent problem. The aims of order acceptance scheduling problem and multi-agent scheduling problem are

similar: they both try to construct settings for more specified conditions. The paper of Reisi and Moslehi (2015) shows one practicable application of combining the order acceptance with multi-agent scheduling problems. In their research, they formulated a mathematical model for their problem and proposed a hybrid genetic algorithm to solve this problem.

Inspired by the work of Reisi and Moslehi (2015), this chapter tries to extend their study in two perspectives: (1) we modify the setting of the problem in their paper and consider a new two-agent and order acceptance scheduling problem; (2) we propose algorithms which apply to both problems. Thus, this chapter focuses on two problems, problem 1 and problem 2. The problems are two-agent single machine order acceptance scheduling problems with different objective functions. The objective function of problem 1 is to maximize the value of total revenues of accepted orders subtracted by weighted lateness of first agent, subject to an upper bound of weighted number of tardy jobs from the second agent. The objective function of problem 2 is to maximize the net profit gained by the sum of revenues of accepted orders minus the weighted tardiness from agent A jobs, meanwhile the upper bound of the weighted number of tardy jobs belonging to B could not be violated. It could be seen that the only difference between the two problems is the penalty functions of late completion of jobs. When jobs are completed after their due dates, both penalty functions take the punishment by decreasing the objective function value. However, for the penalty function of the first problem (weighted lateness), there are rewards for early completion of agent A jobs, which doesn't apply to that of the second problem.

The scheduling problem considered in this chapter could be applied to the following scenario. A restaurant provides service to two kinds of customers: (1) customers who dine in the restaurant, (2) customers who order take-out services. Jobs in this scheduling problem are the dishes to be served to the customers. During the lunch and dinner time, the restaurant could not accept all the

orders. Thus, it needs to reject some of the orders based on profitability of the orders. The objective of the restaurant is to maximize the profit, meanwhile the weighted number of tardy jobs from take-out-customers could not exceed the upper bound. To solve these problems, one meta-heuristic (Particle Swarm Optimization) and one heuristic are proposed. The rest of the chapter is organized as follows: related work will be reviewed in section 3.2, detailed description of the problems and problem formulation will be presented in section 3.2 and 3.3, the description of the proposed algorithms in this chapter will be provided in section 3.4, numerical results will be presented in section 3.5, and in section 3.6, there is a summarization of this chapter.

3.2. Literature review

Guerrero and Kern (1988) are pioneers in the study of order acceptance problems, they pointed out the importance of selecting proper number of orders instead of accepting orders without any consideration. Pourbabai (1989) first developed a mathematical model to help the manufacturers select the orders to select among all the orders. Slotnick and Morton (1996) worked an order acceptance problem in a single machine environment, of which objective is the revenues minus weighted lateness penalties. They proposed an exact algorithm (branch and bound), and two heuristics to solve the problem. Ghosh (1997) extended the work of Slotnick and Morton by providing the NP-hardness proof of the problem in the former paper, in addition, they also provided two pseudo-polynomial algorithms to solve the problem, and a fully-polynomial-time approximation scheme. Lewis and Slotnick (2002) worked on a multi-period order acceptance scheduling problem: decisions on whether accepting the orders are made through different periods. In their problem, rejecting jobs leads to the loss of customers, and the profitability of decisions is examined at each stage. They proposed a dynamic programming algorithm and some

heuristics in their paper. Slotnick and Morton (2005) considered another order acceptance scheduling problem by replacing the penalty function with weighted tardiness. Rom and Slotnick (2009) applied genetic algorithm to solve the order acceptance scheduling problem with tardiness revenues.

Many researchers added variants to order acceptance scheduling problem. Charnsirisaksul et al. (2004) studied a single machine order acceptance problem where pre-emption is allowed. Oguz (2010) considered sequence-dependent set-up times in their problem and arbitrarily assigned deadlines to the orders. Zhong and Ou (2012) added the variant of machine availability to order acceptance scheduling problems, in which jobs could only be processed during specific time intervals due to the availability of machines, when the workload is heavier than the capacity, manufacturers should decide either to reject or outsource some of the orders. Xiao et al. (2012) worked on an order acceptance scheduling problem in permutation flow shop environment, they formulated their problem into an integer programming problem and applied simulated annealing based on partial optimization to solve the problem. Lei and Guo (2015) considered an order acceptance problem in work shop environment, they formulated this problem into a mixed integer programming problem and applied parallel neighborhood search algorithm.

There are some studies on order acceptance scheduling with bi-criteria objectives. Lei and Guo (2015) considered a problem of which the objective is to maximize total net revenue and minimize makespan simultaneously. Ou and Zhong (2017) worked on a bi-criteria order acceptance scheduling problem, with upper bounds on the number of the rejected job as well as the total processing time of accepted jobs. Reisi and Moslehi (2015) integrated two-agent scheduling into order acceptance scheduling problem: there are two agents in their problem, with the objective to maximize the total revenues of accepted orders minus the total weighted lateness

of accepted jobs from first agent. In addition, there is an upper bound of weighted number of tardy jobs from second agent. They formulated it as an integer programming problem and applied hybrid genetic algorithm to solve this problem.

3.3. Problem definition and notations

In the problems of this chapter, there are two agents, agent *A* and agent *B*, each of the agent has a job set. The jobs from these two job sets are assigned with given processing time, weight, due date and revenue associated with them. All the jobs are released at time zero, and pre-emption is not allowed. All the jobs are processed on a single machine. This chapter tries to maximize the objective function of agent A subject to an upper bound of agent B. Following notations are used to describe this problem.

n^A/n^B	number of jobs in agent *A*/ agent *B*
n	total number of jobs, $n = n^A + n^B$
l	index for A-jobs
m	index for B-jobs
j	index used to denote job set;
i	index used to denote jobs in job set j
J^A/J^B	job set belonging to agent A/ agent B, $J_A = \{J_1^A, J_2^A, J_3^A, \ldots J_{n^A}^A\}$, $J_B = \{J_1^B, J_2^B, J_3^B, \ldots J_{n^B}^B\}$
P_i	processing time of job i

D_i	due date of job i
W_i	weight of job i in job set J
R_i	revenue of job i in job set J
Y_i	decision variable of job acceptance, $\begin{cases} \text{if job i is accepted, } Y_i = 1 \\ \text{else } Y_i = 0 \end{cases}$
σ	ordered set of accepted jobs already scheduled
$C_i(\sigma)$	completion time of accepted job i under σ
U_i	index used to denote whether job is tardy, $\begin{cases} \text{if } C_i(\sigma) > D_i, U_i = 1 \\ \text{else } U_i = 0 \end{cases}$
T_i	tardiness variable for job i, $\begin{cases} \text{if } C_i(\sigma) > D_i, T_i = C_i(\sigma) - D_i \\ \text{else } T_i = 0 \end{cases}$
L_i	lateness variable for job i, $L_i = C_i(\sigma) - D_i$
Q	upper bound of wegithed number of tardy jobs of agent B

Based on the notations above, problem 1 could be written as 1|OA, $\sum_{1\le i\le n^B} w_i^B U_i^B \le Q|\sum_{1\le i\le n} R_i Y_i - \sum_{1\le i\le n^A} W_i L_i$. The term OA is used for order acceptance. Problem 1 is a single machine scheduling problem with the objective of maximizing the net profit gained by the sum of revenues from accepted orders minus the total weighted lateness from accepted agent A orders, subject to an upper bound of total weighted number of tardy jobs from agent B.

Similarly, problem 2 could be written as 1|OA, $\sum_{1\le i\le n^B} w_i^B U_i^B \le Q|\sum_{1\le i\le n} R_i Y_i - \sum_{1\le i\le n^A} W_i T_i$. The penalty function in this problem is replaced by the total weighted tardiness of accepted agent A jobs instead of total weighted lateness.

Computational complexity of problem 1 is proved to be NP-hard by Reisi and Moslehi (2015), they showed that the problem 1 (1|OA, $\sum_{1\le i\le n^B} w_i^B U_i^B \le Q| \sum_{1\le i\le n} R_i Y_i - \sum_{1\le i\le n^A} W_i L_i$) is a special case of $1|\sum_{1\le i\le n^B} w_i^B U_i^B \le Q| \sum_{1\le i\le n^A} W_i L_i$, therefore, problem 1 is also an NP-hard problem. Problem 2 reduces to $1| |\sum_{1\le i\le n^A+n^B} W_i T_i$ when Q is set to infinite and R_i are set to zero. Since $1| |\sum_{1\le i\le n^A+n^B} W_i T_i$ is NP-hard, problem 2 is also an NP-hard problem.

3.4. Problem formulation

Reisi and Moslehi (2015) formulated the problem 1 into an integer programming mathematical model. Based on their formulation, this chapter developed an integer programming model for problem 2.

When formulating the mathematical model for problem 2, a new decision variable Z_{ij} is introduced, which denotes the relative position between i and j. The variable Z_{ij} takes the value of 1 if job i precedes j in the schedule, and it equals to 0 in other situations. Note that the precedence here is not the immediate precedence (i.e., Z_{ij} takes the value of 1 doesn't mean that job i and job j are adjacent).

By means of the notations above, problem 2 could be formulated as follows:

Max $\sum_{1\le i\le n} R_i Y_i - \sum_{n1+1\le i\le n} W_i T_i$ (3.1)

s.t

$Y_i+Y_j <= 1 + Z_{ij}+Z_{ji}$ $i, j \in J, i< j$ (3.2)

$Y_i+Y_j>=2(Z_{ij}+Z_{ji})$ $i, j \in J, i< j$ (3.3)

$Z_{ik}+Z_{kj} + Z_{jk}<=2$ $i, j, k \in J, (i \ne j, j \ne k, i \ne k)$ (3.4)

T_i=$\sum_{1\le j\le n} P_j\, Z_{ji}$+$P_i Y_i - D_i$ $\quad n^A + 1 \le i \le n$ (3.5)

$T_i \ge 0$ $\quad n^A + 1 \le i \le n$ (3.6)

$\sum_{1\le j\le n} P_j\, Z_{ji}$+$P_i Y_i$ <= $D_i Y_i + BigMU_i$ $\quad n^B + 1 \le i \le n$ (3.7)

$\sum_{n_1+1\le j\le n} W_i\, U_i$ <= Q_2 $\quad n^B + 1 \le i \le n$ (3.8)

U_i, binary variable, $n^B + 1 \le i \le n$ (3.9)

Y_i, binary variable, $1 \le i \le n$ (3.10)

Z_{ij}, binary variable, $1 \le i \le n, 1 \le j \le n$ (3.11)

Equation (3.1) represents the objective function of problem 2: total revenues of accepted orders minus the weighted tardiness of agent B orders. Constraint (3.2), (3.3) and (3.4) determines the relative positions of jobs. Constraint (3.2) requires that a job must precede another if both jobs are accepted. Constraint (3.3) sets the value of Z_{ij} or Z_{ji} to zero when job i or j is rejected, preventing the situation that Z_{ij} and Z_{ji} both are equal to zero. Constraint (3.4) ensures that there are no job loops in the sequence. Job loop here represents a situation in which job i precedes job j, job j precedes job k, job k precedes job i, which is logically wrong. Constraint (3.5) and (3.6) determines the value of tardiness variable for job i, constraint (3.7) determines the value of U_i. Constraint (3.8) ensures that the upper bound of weighted number of tardy jobs from agent B will not be exceeded. Constraint (3.9), (3.10) and (3.11) are the binary constraints for the corresponding variables.

3.5. Proposed algorithms

In this section, algorithms including a heuristic and PSO are proposed to solve the problems in this chapter.

3.5.1. Heuristic

The heuristic in this section is based on dispatching rules of *WSPT* (weighted shortest processing time) rule and *EDD* rule. In this heuristic, accepted jobs from first agent are arbitrarily sequenced at the beginning of the schedule and are sorted in *WSPT* order, accepted jobs from the second agent are sequenced after the last accepted job from first agent. Within the set of accepted jobs from the second agent, there are two sub sets: early set and late set. The jobs are sequenced in *EDD* order in early set, and the tardy jobs are sequenced at the end of schedule after the early set. Following are the steps of the proposed heuristic.

Step 1: Determine the sequence for accepted jobs from first agent

> Step 1.1: Order the jobs from first agent in *WSPT* order, the jobs ranked in a non-decreasing order of the P_i/W_i ratio.
>
> Step 1.2: Select the jobs to be accepted: if $R[i]>W[i]*(C_i(\sigma) - D[i])$, accept the job; else, reject the job.

Step 2: Determine the sequence for accepted jobs from second agent

> Step 2.1: Order the jobs from the second agent in *EDD* order: jobs are ranked in an non-decreasing order of their due dates.

Step 2.2: Determine the early set and late set: if $C_i(\sigma)$ + $P[i]$> $D[i]$, select this job into early set; otherwise, put it into the late set.

Step 3: Make the solution feasible. This step is only conducted when the weighted number of tardy jobs from agent B in current solution exceeds the given upper bound.

Step 3.1: re-order the tardy jobs from agent B in non-decreasing order of $R[i]/W[i]$ ratio.

Step 3.2: Reject the jobs from the set one by one until the weighted number of tardy jobs is lower than the upper bound Q.

3.5.2. Particle Swarm Optimization

Though PSO is applicable to many scheduling problems, there are some challenges when using it. The main challenge for this algorithm is to design a solution representation mechanism (Sahu et al., 2017) for the problem. In other words, a mechanism establishing the rules to interpret the data of particles into actual schedule. Furthermore, additional efforts are required to adjust to order acceptance scheduling problems, because there are two sub-problems within the order acceptance scheduling problems. The first sub-problem is to determine the job set for accepted job set, another sub-problem is to sequence those jobs from the job set.

Therefore, a PSO algorithm containing parallel parameters of position values and velocities is proposed. Since it is difficult for the classical PSO with only one group of parameters to solve the problems in this chapter, two groups of position values and velocities are embedded in the algorithm, the first group focuses on the selection of accepted jobs, and another group determines the sequence of jobs with the given accepted job set. There are n dimensions in this PSO in total, representing number of the particles, each dimension represents a solution. $X1_j$ and $X2_i$ denote

the position value related to order acceptance decision making and job sequencing respectively, solutions are generated based on these parameters, and the parameters are updated after each iteration. Following are the notations to describe the proposed PSO algorithm.

m Number of particle,

k Number of iterations.

$x1_{ij}^{k}$ Position value of i th particle of j th dimension in k th iteration related to order acceptance.

$X1_{i}^{k}$ Position values of i th particle in k th iteration related to order acceptance,

$v1_{ij}^{k}$ Velocity of i th particle of j th dimension in k th iteration related to order acceptance.

$V1_{i}^{k}$ Velocity of i th particle in k th iteration related to order acceptance.

$x2_{ij}^{k}$ Position value of i th particle of j th dimension in k th iteration related to job sequencing.

$X2_{i}^{k}$ Position values of i th particle in k th iteration related to job sequencing,

$v2_{ij}^{k}$ Velocity of i th particle of j th dimension in k th iteration related to job sequencing.

$V2_{i}^{k}$ Velocity of i th particle in k th iteration related to job sequencing.

pb_{ij}^{k} Personal best position value in k th iteration related to job sequencing

gb_{ij}^{k} Global best position value of in k th iteration related to job sequencing

c_1, c_2 Social cognitive numbers

w Inertia weight

The steps of proposed PSO algorithm are as follows:

Step 1: Initialize the parameters of velocity and position value for each particle.

Step 1.1: Set *k*=0, *m*= 50.

Step 1.2: Generate initial values for $X1_i^0,\ V1_i^0,\ X2_i^0,\ Z2_i^0$ randomly.

Step 1.3: Use the Sequence Generator to generate actual schedule.

Step 1.4: Initialize the position value vector and velocity vector of the personal best solutions and the global best solution related to order acceptance and job sequencing respectively.

Step 2: Do the following for each particle:

Step 2.1: *k*=*k*+1

Step 2.2: Update velocity and position value using the formulas given below:

Step 2.2.1: Update order-acceptance-related vectors:

$$\mathrm{v1}_{ij}^{k} = w^{k-1}\mathrm{v1}_{ij}^{k} + c_1 * r_1 * z_1 + c_2 * r_2 * z_2;$$

$$x1_{ij}^{k} = x1_{ij}^{k} + \mathrm{v1}_{ij}^{k};$$

Step 2.2.2: Updae sequencing-related vectors:

$$\mathrm{v2}_{ij}^{k} = w^{k-1}\mathrm{v2}_{ij}^{k} + c_1 * \mathrm{r1} * (pb_{ij}^{k} - x2_{ij}^{k}) + c_2 * r_2 * (gb_{j}^{k} - x2_{ij}^{k});$$

$$x2_{ij}^{k} = x2_{ij}^{k} + \mathrm{y}_{ij}^{k};$$

Step 2.3: Normalize the velocity and position values between [-4,4].

Step 2.4: Use the Sequence Generator to generate actual schedule.

Step 2.5: Apply local search schemes

Step 2.6: Update the position values and velocities of personal best and global best.

Step 3: Stopping criterion.

If the number of iterations exceeds the maximum number of iterations, then stop; otherwise, go back to step 2.

Parameter initialization

At the beginning, parameters are generated with the following equations:

$$x1_{ij}^{0} = X1_{min} + (X1_{max} - X1_{min}) * U(0,1)$$

$$v1_{ij}^{0} = V1_{min} + (V1_{max} - V1_{min}) * U(0,1),$$

$$x2_{ij}^{0} = X2_{min} + (X2_{max} - X2_{min}) * U(0,1)$$

$$v2_{ij}^{0} = V2_{min} + (V2_{max} - V2_{min}) * U(0,1),$$

Where $X1_{max}$, $V1_{max}$, $X2_{max}$, $V2_{max}$ are equal to 4; $X1_{min}$, $V1_{min}$, $X2_{min}$, $V2_{min}$ are equal to -4. $U(0,1)$ is a random number within the range of [0,1]. There, all these parameters are within the range of [-4,4].

Sequence generator

As mentioned above, the sequence generating process in the proposed PSO consists of two steps, the first step is to select the accepted jobs from the whole job set, and the second step is to sequence the accepted jobs.

Step 1: Select the jobs to accept from job set J.

Decisions on order acceptance are made according to the values of $X1$, job j in particle i could only be accepted when $x1_{ij}^{k}$ is greater than 0.

Step 2: Sequence the accepted jobs.

Jobs are sequenced according to the Minimum position value (MPV) rule, which means that jobs are ordered in a non-decreasing order of their position values.

Updating parameters

Parameters are updated at the end of iterations. Two different ways to update parameters are applied to update the values of $X1$, $X2$ respectively.

Update order acceptance parameters of $X1$

$$v1_{ij}^{k} = w\, v1_{ij}^{k-1} + c_1 * r_1 * z_1 + c_2 * r_2 * z_2$$

$$x1_{ij}^{k} = x1_{ij}^{k-1} + v1_{ij}^{k};$$

Parameters of z_1 and z_2 are variables representing the status of job j : variable z_1 takes the value of 1 when job j in i^{th} particle is accepted in personal best solution in k^{th} iteration, otherwise, it takes the value of -1. Likewise, value of variable z_2 is determined by whether the job is in accepted in global best solution. Parameters of c_1 and c_2 are social coefficient quantifying the extent the individual relies on its own experiment (personal best) or global best. Inertia w is the

resistance of any physical object to any change in the state of motion, it controls velocity and direction of all the particles. Variables of r_1 and r_2 are random numbers within the range of [0,1]. In this chapter, $c_1 = 0.9,\ c_2 = 0.9,\ w = 0.9$.

Update order parameters of $X2$

$$v2\ {}_{ij}^{k} = w\ v2_{ij}^{k-1} + c_1 * r_1 * (pb_{ij}^{k-1} - x2_{ij}^{k-1}) + c_2 * r_2 * (gb_{ij}^{k-1} - x2_{ij}^{k-1})$$

$$x2_{ij}^{k} = x2_{ij}^{k-1} + v2\ {}_{ij}^{k};$$

The method to update parameters of $X2$ is closer to the original method in classical PSO, where $c_1,\ c_2$ and w also take the value of 0.9.

Local search

Two kinds of local search schemes (local search 1, local search 2) are proposed to improve the solution of each particle.

Local search 1 tries to improve the solution by inserting the accepted jobs in the positions with the maximum objective function, whereas local search 2 focuses on attempting to insert rejected jobs into the sequenced schedule. As mentioned above, for these two schemes, jobs are selected from different sets, these schemes try to improve the objective function by iteratively inserting the jobs from the given set into the best positions. Following is an general explanation on the steps of the local search schemes above.

Step 1: Select the first job in the given set and move it to the end of the current accepted schedule.

Step 2: Insert the selected job into every possible position and calculate the objective function for resultant sequence. Select the best position for the job inserted.

Step 3: If there is an improvement of the objective function, update the schedule and objective function, go to step 1; if there is no improvement, put the selected job into its initial position, select the next job in the set and return to step 2 until the last job in the sequence is selected. Stopping criterion is satisfied if there is no improvement after going through all the jobs in the given set.

Normalization of parameters

The absolute values of parameters tend to be very large after many iterations, leading to the solutions stuck in local optimum. Thus, parameters are normalized after each iteration. Parameters exceeding the upper limit take the value of 4, and the parameters that are less than the lower limit take the value of -4.

3.6. Numerical results

This section evaluates the performance of the proposed algorithms on two problems. Numerical analysis is provided based on solution and CPU time of the proposed algorithms. The proposed algorithms (PSO algorithm and heuristic algorithm) in this chapter are coded in C++ programming language, the numerical experiments are implemented in a computer environment with AMD Opteron 2.3GHz with 256 GB RAM on Windows 7. The linear-programming-based mathematical model proposed for problem 2 is solved by AMPL software with CPLEX solver, in computer system of iMac desktop with 3.3GHz 8 GB RAM.

There are three sub-sections within the section of numerical results. The first sub-section explains the settings of the instances used in this chapter. The second sub-section aims to test the robustness of proposed problems on problem 1 by comparing them with the results in Reisi and

Moslehi (2015). The third sub-section tries to evaluate these algorithms on a new problem (problem 2).

Following are notations used to report and evaluate the numerical results.

CPLEX: Value of objective function generated by the mathematical model.

ABS: Absolute value of objective function generated by algorithms.

ABS_{best}: *ABS* value of the objective function generated by the algorithm which performs the best.

APD: Absolute percentage of deviation, $APD = (ABS - CPLEX)/CPLEX * 100\%$

RPD: Relative percentage of deviation, $RPD = (ABS - ABS_{best})/ABS_{best} * 100\%$

CPU: Running time of proposed algorithm.

In addition, Avg, Min, Max are notations representing average, minimal, maximal values of corresponding results respectively.

3.6.1. Instance setting

In this chapter, instances generated in the study of Reisi and Moslehi (2015) are recognized as the benchmark instances, different algorithms are evaluated based on their performance on these benchmark instances.

The setting of the benchmark instances is as follows: processing times and weights are generated randomly from a uniform distribution within the interval of [1,10]. When generating due dates, two additional parameters are introduced: tardiness factor τ, range factor r, due dates are generated randomly within the range of [*Total_processing_time*$*(1 $-\tau - r/2$)),

Total_processing_time$*(1-\tau - r/2)]$, in which the *Total_processing_time* denotes the sum of processing times from all the jobs. The revenues are generated from a uniform distribution within the interval of [1, 2*p_i], upper bound *Q* is generated within the range of [0, $\sum_{n_1+1\leq i\leq n} W_i$]. The instances are sorted into eight groups according to the parameter setting, the sizes of instances ranging from 20 to 150. In the first four groups, $n_1 = 2n_2$, in the remaining groups, $2n_1 = n_2$. Parameter of τ takes the value of 0.3 in the Groups of G01, G02, G05, G06, and it takes the value of 0.7 in the groups of G03, G04, G07, G08. Parameter of *r* takes the value of 0.2 in groups of G01, G03, G05, G07. In the benchmark instances, there are 1280 instances in total, and 20 instances for each parameter setting. The numerical analysis is mainly based on the average results obtained by different results in different parameter settings.

3.6.2. Numerical analysis for problem 1

As mentioned above, performance of PSO and heuristic are compared with the results in Reisi and Moslehi (2015). Thus, numerical results of GA and *CPLEX* solution in Reisi and Moslehi (2015) are obtained from the authors. In small instances, performance of these algorithms along with the GA algorithm are compared with the *CPLEX* solution, while in large instances, the performance of PSO, heuristic and GA is compared with each other. Note that the CPU time of heuristic is shown in the corresponding tables since the proposed heuristic algorithm could solve all instances within 0.1 seconds.

Table 3.1 displays the computational results of integer programming model. According to the results, it could be seen that their model could solve problem instances with up to 60 jobs. It could also be seen that CPU time of G03 and G07 is significant longer than other groups, when

$\tau = 0.3$, r $= 0.6$. Both the maximum CPU time in these groups exceed 1000 seconds, indicating that it is more difficult for the system to search for the optimal solutions in these groups.

Table 3.1 CPU time of CPLEX solutions in Problem 1

Group	n	CPU			Group	n	CPU		
		Avg	Min	Max			Avg	Min	Max
G01	20	0.09	0.02	0.98	G05	20	0.06	0.02	0.4
	30	0.37	0.07	3.27		30	1.45	0.14	12.28
	40	1.5	0.18	7.53		40	1.16	0.16	3.85
	60	1.51	0.41	15.63		60	27.34	0.76	201.91
G02	20	0.03	0.02	0.15	G06	20	0.04	0.02	0.11
	30	0.08	0.06	0.1		30	0.4	0.13	1.93
	40	0.22	0.17	0.3		40	0.22	0.12	0.64
	60	0.4	0.31	0.64		60	1.77	0.58	8.1
G03	20	0.39	0.02	1.18	G07	20	0.32	0.08	0.98
	30	4.3	0.09	15.07		30	4.51	0.55	16.78
	40	28.78	0.24	121.75		40	35.92	0.48	200.73
	60	308.44	0.53	1314.29		60	519.62	23.01	1442.53
G04	20	0.07	0.02	0.64	G08	20	0.06	0.02	0.48
	30	0.34	0.07	1.46		30	0.43	0.08	1.88
	40	1	0.17	4.59		40	0.83	0.17	3.21
	60	11.7	0.48	138.64		60	7.33	0.59	88.78

Table 3.2 displays the APD values of the three algorithms in problem 1. As shown in the table, the PSO algorithm performs best in small instances in terms of solution quality: solutions of PSO is only 0.01 percent away from the CPLEX solutions. In addition, the maximal APD of PSO is lower than 0.1%, which is also lower than other algorithms. On the other hand, though the heuristic deviates furthest from CPLEX solutions, the average APD of it is lower than 1.5%, indicating the effectiveness of the heuristic in small instances.

Table 3.2 Comparison of algorithms in problem 1 in small instances

Group	n	GA			Heuristic		PSO		
		Average ABS	Average APD	Average CPU time	Average ABS	Average APD	Average ABS	Average APD	Average CPU time
G01	20	2518.30	0	0.94	2517.60	0.03	2518.35	0	0.56
	30	5682.55	0.02	1.95	5680.75	0.05	5683.30	0	1.96
	40	9663.80	0.01	3.9	9658.3	0.06	9664.60	0	5
	60	21319.3	0.01	8.09	21317	0.02	21321.25	0	17.75
G02	20	2460	0.01	1.01	2458.55	0.02	2460.25	0	0.6
	30	5166	0	1.66	5165.15	0	5166.10	0	2.12
	40	9784.05	0	3.54	9784.05	0	9784.40	0	5.38
	60	21093	0	9	21093.15	0	21093.15	0	19.13
G03	20	920.50	0.11	1.94	904.70	2.25	921.30	0.03	0.47
	30	1951.60	0.21	3.76	1940.90	0.7396	1954.40	0.06	1.89
	40	3386.15	0.33	7.01	3370.60	0.81	3394.65	0.05	4.58
	60	7149.85	0.34	15.82	7119.25	0.749	7168.10	0.07	16.25
G04	20	927.95	0.27	1.41	921.25	1.11	929.90	0.01	0.51
	30	2104.65	0.24	3.01	2067.50	2.019	2109.15	0.04	1.79
	40	3606.20	0.26	6.16	3555.45	1.81	3616.10	0.02	4.59
	60	8228.95	0.11	13.44	8108.20	1.56	8236.50	0.02	16.62
G05	20	3945.65	0.04	1.74	3943.60	0.09	3947.25	0	0.79
	30	9043.75	0.02	4.44	9034.70	0.13	9045.20	0	2.90
	40	16682.30	0.03	8.84	16683.30	0.03	16687.75	0	7.73
	60	35684.45	0.03	20.79	35681.90	0.03	35684.45	0	27.86
G06	20	4051.50	0.09	1.67	4035.35	0.44	4052.10	0	1.07

	30	9087.05	0.02	3.36	9045	0.49	9088.50	0	4.19
	40	15829.60	0	6.10	15762.45	0.44	15830.40	0	10.50
	60	36173.25	0	13.8	35997.25	0.5	36174.90	0	37.70
G07	20	1179.8	0.16	1.70	1145.80	3.20	1181.65	0	0.38
	30	2422.85	0.97	3.73	2335.45	4.52	2446.05	0.02	1.41
	40	5065.50	0.69	7.16	5096.70	2.53	4971.40	0.08	3.60
	60	10551.75	1.17	19.06	10276.80	3.77	10674.25	0.02	13.17
G08	20	1414.75	0.02	0.88	1361.90	4.16	1414.90	0	0.40
	30	3572.65	0.04	2.76	3422.10	4.65	3573.50	0	1.54
	40	6018.05	0.13	5.04	5747.70	4.62	6026	0	4.02
	60	13176.25	0.05	11.7	12704.15	3.66	13180.60	0.01	14.58
Average		8745.69	0.17	6.27	8685.52	1.39	8750.01	0.01	7.22

Table 3.3 shows the results of numerical experiments in problem 1. From the table it could be seen that PSO also outperforms the algorithms of GA and heuristic in large instances: the *RPD* of PSO is lower than those of other algorithms with only few exceptions; average *RPD* of PSO in all groups is only 0.05, indicating the robustness of PSO on this problem. Nonetheless, observation from the perspective of CPU time somehow provides a different picture: average CPU time of PSO is 183.54 seconds, which is longer than two times as many as that of GA; on the other hand, CPU time of both meta-heuristics (GA and PSO) increases significantly as the size of instances increases, and growth of CPU time for PSO is more remarkable. Thus, when the size of instances to be solved larger than the upper bound 150 in this chapter, performance of heuristic algorithm might be more prominent due to its efficiency. Besides, it is noteworthy to point out the average value and the maximal value of *RPD* of heuristic is 1.21 percent and 4.61 percent respectively, which also justify the heuristic.

Table 3.3 Comparison of algorithms in problem 1 in large instances

Group	n	GA	Heuristic	PSO

		Average ABS	Average RPD	Average CPU time	Average ABS	Average RPD	Average ABS	Average RPD	Average CPU time
G01	80	39999.70	0	18.36	39996.55	0.01	40001.35	0	47.17
	100	60348.05	0	33.35	60344.20	0.01	60351	0	98.50
	120	90745.90	0	53.99	90746	0	90747.45	0	175.65
	150	136730.50	0	95.76	136730.25	0	136732.45	0	349.23
G02	80	40321.55	0	19.09	40321.80	0	40322.25	0	50.24
	100	59533.70	0	35.84	59533.25	0	59534.45	0	106.82
	120	88112.60	0	55.2	88112.85	0	88113.20	0	193.73
	150	135944.40	0	93.05	135945.20	0	135945.20	0	375.07
G03	80	13218.35	0.18	33.01	13169.90	0.57	13241.65	0	43.49
	100	21454.10	0.10	51.24	21390.30	0.41	21474.45	0	90.17
	120	29716.55	0.24	75.88	29621.80	0.56	29772.05	0.04	154.81
	150	45783.90	0.17	144.39	45646.40	0.47	45850.05	0.02	333.35
G04	80	16005.40	0.08	27.18	15764.05	1.63	16017	0	43.59
	100	23260.65	0.13	43.34	22756	2.39	23290.40	0	86.89
	120	33656.80	0.11	64.35	33128.60	1.77	33688.35	0.02	148.34
	150	55188.40	0.03	113.93	54435.75	1.40	55145.45	0.13	297.54
G05	80	65441.85	0.02	41.28	65444.25	0.02	65454.45	0	72.36
	100	105339.85	0.01	69.83	105337.10	0.01	105351.90	0	152.88
	120	144242	0.01	130.24	144235.25	0.01	144246.15	0	273.69
	150	224901.75	0.01	201.08	224900.70	0.01	224919.10	0	565.53
G06	80	67786.35	0	27.94	67596.85	0.30	67786.15	0	95.73
	100	102016.45	0	58.30	101679.35	0.33	102016.30	0	200.17
	120	142779.90	0	93.90	142188.55	0.41	142761.25	0.02	332.17
	150	229024.20	0	151.58	228191.40	0.36	228882.65	0.06	663.42
G07	80	18552.30	1.46	48.56	18146.20	3.59	18823.95	0	11
	100	31254.30	2.01	94.25	31044.10	2.69	31896.40	0	64.70
	120	44185.65	2.16	154.37	43879.85	2.88	45117.95	0.10	116.44
	150	66529.35	1.99	327.84	66258.80	2.39	67843.05	0.04	240.82

G08	80	23772	0.02	24.75	22833.25	4.07	23770	0.03	36.53
	100	36552.20	0.13	41.10	34912.65	4.67	36548.90	0.10	67.50
	120	55435.10	0	57.59	53400	3.77	55273.10	0.30	123.48
	150	85601.85	0	104.03	84987.10	3.98	82243.20	0.74	262.19
Average		72919.86	0.28	80.77	72583.70	1.21	72911.29	0.05	183.54

Based on the discussion above, it could be concluded that both PSO and heuristic could provide reliable solutions which have several advantages: PSO provides better solutions than heuristic and the GA algorithm, while the heuristic could solve the problem in shorter time.

3.6.3. Numerical analysis for problem 2

Table 3.4 CPU time of CPLEX solutions in problem 2

Group	n	CPU			Group	n	CPU		
		Avg	Max	Min			Avg	Max	Min
G01	20	202.06	3001.29	0.29	G05	20	2089.13	38991.2	0.19
G02	20	7.46	117.49	0.18	G06	20	9.69	156.75	0.18
G03	20	75.50	428.41	0.20	G07	20	75.60	448.80	2.80
G04	20	5.19	27.88	0.13	G08	20	9.53	49.89	0.71

Table 3.4 shows the CPU time for the mathematical model of problem 2. As shown in the table, the mathematical model formulated in this chapter could only solve instances with the size up to 20 jobs, which might be due to the computational complexity of the problem. In addition, CPU time of the mathematical model is longer than other settings when tardiness factor takes the value of 0.3 instead of 0.7. In G05, average CPU time to get the CPLEX solutions is 2089.13

seconds, for an instance belonging to this group, it takes more than ten hours (38991.2 seconds) to find the solution.

Table 3.5 Comparison of algorithms in problem 2 in small instances

Group	n	Heuristic		PSO		
		Average ABS	Average APD	Average ABS	Average CPU time	Average APD
G01	20	116.80	0.68	117.60	0.30	0
G02	20	112.50	2.29	115.15	0.35	0
G03	20	98.30	14.21	112.85	0.53	0.14
G04	20	96.75	11.46	108.50	0.61	0.10
G05	20	110.45	6.02	117.45	0.46	0
G06	20	107.15	10.24	119.50	0.53	0
G07	20	71.30	23.57	92.60	0.35	0.23
G08	20	70.20	29.53	98.15	0.49	0.44
Average		97.93	12.25	110.23	0.45	0.11

Table 3.5 displays the results of PSO and heuristic in small instances. As shown in the table, PSO outperforms the heuristic in all groups in terms of objective value, it deviates from the CPLEX solutions only by 0.11 percent. In contrast, solution quality of heuristic is not as good as PSO, the average APD values from five groups (G03, G04, G06, G07, G08) exceed 10 percent. Though it appears that the heuristic in this chapter does not perform as well as in problem 2 due to the high APD values in small instances. It is noteworthy to mention that the heuristic could still be used as a good comparison for PSO due to its efficiency. What's more, the limited

capability of the formulated mathematical model determines that the evaluation of an algorithm relies heavily on the numerical results in larger instances.

Numerical results of PSO and heuristic in problem 2 is presented in Table 3.6, PSO still shows its superiority over heuristic when comparing their solution. However, it could be seen that the gap between their performance is controlled within a reasonable level (30 percent), except in G07 and G08, RPD value in most of the groups is lower than 15 percent. Furthermore, it is shown that performance does not improve as instance size grows.

Table 3.6 Comparison of algorithms in problem 2 in large instances

n	Group	Heuristic		PSO		
		Average ABS	Average RPD	Average ABS	Average CPU time	Average RPD
30	G01	173.75	1.96	176.95	0.89	0
40		225.55	3.05	232.60	2.45	0
60		344.05	1.29	348.60	6.35	0
80		463.15	1.10	468.50	17.24	0
100		608.30	1.20	615.50	32.93	0
120		726.05	0.23	727.70	52.49	0
150		892.95	0.23	894.95	105.01	0
30	G02	169.90	0.61	170.85	0.90	0
40		239.55	0.15	239.90	2.20	0
60		363.25	0	363.25	6.94	0
80		485.35	0.09	485.80	17.87	0
100		601.40	0.19	602.60	40.77	0
120		723.30	0.05	723.65	64.81	0
150		903.70	0	903.70	113.56	0
30	G03	146.90	11.37	164.65	1.86	0
40		204.55	9.94	225.35	4.10	0
60		292.95	10.14	323.45	13.87	0
80		414.80	7.53	447.30	36.97	0

100		537.10	7.21	575.75	70.42	0
120		581.60	8.15	625.75	122.50	0
150		807.35	5.61	852.75	247.89	0
30	G04	146.40	14.27	169.85	2.41	0
40		200.45	9.95	221.75	5.64	0
60		308.90	9.59	341.15	18.73	0
80		419.95	8.50	458	53.93	0
100		549.25	8.75	601	114.19	0
120		633.95	10.49	705.30	203.20	0
150		753.05	11.06	843.50	371.27	0
30	G05	170.25	6.08	180.75	1.58	0
40		234.40	3.34	242.40	3.72	0
60		346.85	4.61	363.10	12.29	0
80		463.80	2.58	476.40	28.42	0
100		590.40	2.65	606.35	54.21	0
120		692.50	3.94	720.15	95.59	0.01
150		853	2.75	877.15	205.54	0
30	G06	171.55	7.68	185.90	1.72	0
40		211.50	7.34	227.20	4.01	0
60		343.10	8.22	373.85	13	0
80		431.15	6.29	460.10	26.34	0
100		570.90	5.90	606.85	55.61	0
120		647.70	8.14	704.20	109.97	0
150		822.70	6.22	877.15	198.15	0
30	G07	114.20	23.30	146.70	1.14	0
40		147.10	23.20	190.40	2.75	0
60		226.35	20.75	284.50	9.13	0
80		297.55	21.60	377.20	20.83	0
100		368.55	20.70	462.10	40.88	0
120		460.10	19.70	570	72.39	0
150		559.60	18.74	685.60	142.20	0
30	G08	131.35	23.49	171.15	1.96	0
40		158.10	26.43	214.60	4.81	0
60		243.80	24.31	321.50	18.36	0
80		326.85	23.24	424.55	50.35	0

100		396	24.61	525.35	101.70	0
120		493.05	25.71	662.35	181.65	0
150		588.45	25.46	788.05	374.11	0
Average		428.18	9.64	468.49	63.64	0

Thus, it could be concluded that PSO could provide better solution. The heuristic can generate solution in shorter time with acceptable percentage deviation.

3.7. Conclusions

This chapter works on two order acceptance two-agent scheduling problems, objective function of first problem is to maximize the total revenues of accepted orders, minus the total weighted lateness of jobs from agent A, and objective function of second problem is to maximize the total revenues of accepted orders, minus the total weighted tardiness of jobs from agent A. Both of the problems includes a constraint that the sum of weighted number of tardy jobs belonging to agent B could not exceed a given upper bound. Algorithms including a meta-heuristic (PSO) and a heuristic are proposed to solve these problems. In addition, a mathematical model for the second problem is provided.

According to the numerical results, it turns out that both of the proposed algorithms are applicable to the problems studied in this chapter, PSO could provide better solutions while the heuristic could provide solutions with acceptable deviation percentage from PSO in shorter time period.

In the future, the mathematical model formulated in this chapter could be improved by increasing its capability, and there is also much work could be done to decrease the CPU time of proposed PSO algorithm.

Chapter 4

Conclusions

In this thesis, three multi-agent single machine scheduling problems were considered. Jobs belonging to two agents need to be sequenced on a single machine, and criterion for each agent should be considered when making decisions. The first problem is a two-agent scheduling problem with the objective to minimize weighted number of tardy jobs from the first agent, subject to an upper bound of weighted number of tardy jobs from the second agent. In the second and the third problem, order acceptance is integrated into two-agent scheduling problem. The objective of the problem is to maximize the benefits brought by the revenues of accepted jobs incurred from late deliveries of orders from the first agent, meanwhile, weighted number of tardy jobs from the second agent could not exceed its upper bound.

For the first problem, two meta-heuristics (PSO, Tabu Search) are applied. In addition, we also proposed an exact algorithm which could solve instances up to 280 jobs. Furthermore, some additional adjustments are added to algorithms of PSO and Tabu Search, and these adjustments are justified by the improvement on solutions. Numerical experiment of this problem indicates that both algorithms could solve the problem effectively, with deviation percentage lower than 5 percent from the optimal solution. Numerical result also shows that Tabu Search outperforms PSO because its solution is better and it could save 90 percent running time compared to PSO.

For the second and the third problem, PSO and a heuristic are proposed to solve this problem. In addition, a mathematical model is formulated to solve the third problem. The numerical results of the second and the third problem show that these algorithms could solve these problems

effectively. PSO provides better solutions than the heuristic in both problems, while the heuristic could solve all the instances in numerical experiments within 0.1 seconds. By proposing algorithms and problem formulation, we tried to extend the study of order acceptance and two-agent scheduling problems.

The two-agent scheduling problem is attracting the attention of many researchers. Some problems have been solved in the last decade, however, there are many two-agent scheduling problems which has not been considered in the literature. In future, study of Chapter 3 could be developed by considering more problems with various penalty functions of agent A: (1) 1|OA, $\sum_{1\le i\le n^B} w_i^B U_i^B \le Q \;\mid\; \sum_{1\le i\le n} R_i Y_i - \sum_{1\le i\le n^A} W_i(L_i + E_i)$, (2) 1|OA, $\sum_{1\le i\le n^B} w_i^B U_i^B \le Q|\sum_{1\le i\le n} R_i Y_i - \sum_{1\le i\le n^A} L_i$, (3) 1|OA, $\sum_{1\le i\le n^B} w_i^B U_i^B \le Q|\sum_{1\le i\le n} R_i Y_i - \sum_{1\le i\le n^A} T_i$, (4) 1|OA, $\sum_{1\le i\le n^B} w_i^B U_i^B \le Q|\sum_{1\le i\le n} R_i Y_i - \sum_{1\le i\le n^A} L_i + E_i$.

References

Agnetis, A., Mirchandani, P. B., Pacciarelli, D., Pacifici, A., Agnetis, A., Pacciarelli, D., & Pacifici, A. (2004). Scheduling Problems with Two Competing Age, *52*(2), 229–242.

Agnetis, A., Billaut, J. C., Gawiejnowicz, S., Pacciarelli, D., & Soukhal, A. (2014). Multiagent scheduling. *Berlin Heidelberg: Springer Berlin Heidelberg. doi*, *10*(1007), 978-3.

Baker, K. R., & Smith, J. C. (2003). A MULTIPLE-CRITERION MODEL FOR MACHINE SCHEDULING, 7–16.

Brewer, Paul J. and Charles R. Plott, (1996). "A Binary Conflict Ascending Price (BICAP) Mechanism For The Decentralized Allocation of the Right to Use Railroad Tracks," International Journal of Industrial Organization, 14:857-886.

Cesaret, B., Oğuz, C., & Sibel Salman, F. (2012). A tabu search algorithm for order acceptance and scheduling. *Computers and Operations Research*, *39*(6), 1197–1205.

Charnsirisakskul, K., Griffin, P. M., & Keskinocak, P. (2004). Order selection and scheduling with leadtime flexibility. *IIE Transactions (Institute of Industrial Engineers)*, *36*(7), 697–707.

Cheng, T. C. E., Cheng, S. R., Wu, W. H., Hsu, P. H., & Wu, C. C. (2011). A two-agent single-machine scheduling problem with truncated sum-of-processing-times-based learning considerations. *Computers and Industrial Engineering*, *60*(4), 534–541.

Cheng, T. C. E., Ng, C. T., & Yuan, J. J. (2006). Multi-agent scheduling on a single machine to minimize total weighted number of tardy jobs. *Theoretical Computer Science*, *362*(1–3), 273–281.

Fatos, X., & Ajith, A. (2008). *Metaheuristics for Scheduling in Industrial and Manufacturing Applications edited by Fatos Xhafa, Ajith Abraham. Studies in Computational Intelligence* (Vol. 14).

Ghosh, J. B. (1997). Job selection in a heavily loaded shop. *Computers and Operations Research*, *24*(2), 141–145.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. H. G. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, *5*(C), 287–326. https://doi.org/10.1016/S0167-5060(08)70356-X

Guerrero, H. H., & Kern, G. M. (1988). How to more effectively accept and refuse orders. *Production and Inventory Management Journal*, *29*(4), 59–63. Retrieved from

Karp, R. M. (1972). Reducibility among combinatorial problems. In Complexity of computer computations (pp. 85-103). Springer, Boston, MA

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. Neural Networks, 1995. Proceedings., IEEE International Conference on, 4, 1942–1948 vol.4.

Lei, D., & Guo, X. (2015). A parallel neighborhood search for order acceptance and scheduling in flow shop environment. *International Journal of Production Economics*, *165*, 12–18.

Leung, J. Y., Li, H., & Pinedo, M. (2005). Order scheduling models: an overview. In *Multidisciplinary scheduling: theory and applications* (pp. 37-53). Springer, Boston, MA.

Lewis, H. F., & Slotnick, S. A. (2002). Multi-period job selection: Planning work loads to maximize profit. *Computers and Operations Research*, *29*(8), 1081–1098.

Li, H., & Gajpal, Y. (2017). Two agents single machine scheduling problem with total completion time and total weighted completion time objectives. Operational Research and Management Science Letters, 1(1), 8-16.

Liao, C. J., Tseng, C. T., & Luarn, P. (2007). A discrete version of particle swarm optimization for flowshop scheduling problems. Computers & Operations Research, 34(10), 3099-3111.

Moore, J. M. (1968). An n job, one machine sequencing algorithm for minimizing the number of late jobs. Management science, 15(1), 102-109.

M'Hallah, R., & Bulfin, R. . (2003). Minimizing the weighted number of tardy jobs on a single machine. *European Journal of Operational Research*, *145*(1), 45–56.

Ng, C. T., Cheng, T. C. E., & Yuan, J. J. (2006). A note on the complexity of the problem of two-agent scheduling on a single machine. *Journal of Combinatorial Optimization*, *12*(4), 386–393.

Pandey, S., Wu, L., Guru, S. M., & Buyya, R. (2010). A Particle Swarm Optimization-Based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments. 2010 24th IEEE International Conference on Advanced Information Networking and Applications, 400–407.

Perez-Gonzalez, P., & Framinan, J. M. (2014). A common framework and taxonomy for multicriteria scheduling problems with interfering and competing jobs: Multi-agent scheduling problems. European Journal of Operational Research, 235(1), 1–16.

Pessan, C., Bouquard, J.-L., & Neron, E. (2008). An unrelated parallel machines model for an industrial production resetting problem. European Journal of Industrial Engineering, 2, 153–171.

Pinedo, M. L. (2016). *Scheduling Theory, Algorithms, and Systems* (5th ed).

Pourbabai, B. (1989). A short term production planning and scheduling model. *Engineering Costs and Production Economics*, *18*(2), 159–167.

Reisi-Nafchi, M., & Moslehi, G. (2015). A hybrid genetic and linear programming algorithm for two-agent order acceptance and scheduling problem. *Applied Soft Computing Journal*, *33*, 37–47.

Rom, W. O., & Slotnick, S. A. (2009). Order acceptance using genetic algorithms. *Computers and Operations Research*, *36*(6), 1758–1767. https://doi.org/10.1016/j.cor.2008.04.010

Sahu, S. N., Gajpal, Y., & Debbarma, S. (2017). Two-agent-based single-machine scheduling with switchover time to minimize total weighted completion time and makespan objectives. Annals of Operations Research, 1-18.

Schultz, D., Oh, S. H., Grecas, C. F., Albani, M., Sanchez, J., Arbib, C., ... & Lombardi, G. (2002, June). A QoS concept for packet oriented S-UMTS services. In IST Mobile and Wireless Telecommunications Summit.

Slotnick, S. A., & Morton, T. E. (2007). Order acceptance with weighted tardiness. *Computers & Operations Research*, *34*(10), 3029-3042.

Soomer, M. J., & Franx, G. J. (2008). Scheduling aircraft landings using airlines' preferences. *European Journal of Operational Research*, *190*(1), 277–291.

Tasgetiren, M. F., Sevkli, M., Liang, Y. C., & Gencyilmaz, G. (2004, September). Particle swarm optimization algorithm for permutation flowshop sequencing problem. International Workshop on Ant Colony Optimization and Swarm Intelligence(pp. 382-389). Springer, Berlin, Heidelberg.

W. H. Wu, Yin, Y., Cheng, T. C. E., Lin, W. C., Chen, J. C., Luo, S. Y., & Wu, C. C. (2017). A combined approach for two-agent scheduling with sum-of-processing-times-based learning effect. Journal of the Operational Research Society, 68(2), 111-120.

Wang, J. Q., Fan, G. Q., Zhang, Y., Zhang, C. W., & Leung, J. Y. T. (2017). Two-agent scheduling on a single parallel-batching machine with equal processing time and non-identical job sizes. European Journal of Operational Research, 258(2), 478-490.

Wu, W. H. (2013). An exact and meta-heuristic approach for two-agent single-machine scheduling problem. *Journal of Marine Science and Technology (Taiwan)*, *21*(2), 215–221.

Xiao, Y. Y., Zhang, R. Q., Zhao, Q. H., & Kaku, I. (2012). Permutation flow shop scheduling with order acceptance and weighted tardiness. *Applied Mathematics and Computation*, *218*(15), 7911–7926.

Yoshida, H., Kawata, K., Fukuyama, Y., Takayama, S., & Nakanishi, Y. (2000). A particle swarm optimization for reactive power and voltage control considering voltage security assessment. IEEE Transactions on power systems, 15(4), 1232-1239.

Zhong, X., Ou, J., & Wang, G. (2014). Order acceptance and scheduling with machine availability constraints. *European Journal of Operational Research*, *232*(3), 435–441.