Spring 2011

Optimization Models and Approximate Algorithms for the Aerial Refueling Scheduling and Rescheduling Problems

Sezgin Kaplan
Old Dominion University

OPTIMIZATION MODELS AND APPROXIMATE ALGORITHMS FOR THE AERIAL REFUELING SCHEDULING AND RESCHEDULING PROBLEMS

by

Sezgin Kaplan

B.S. May 2000, Bilkent University, Turkey
M.A. May 2004, Hacettepe University, Turkey
M.S. November 2007, Gazi University, Turkey

A Dissertation Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

ENGINEERING MANAGEMENT

OLD DOMINION UNIVERSITY
May 2011

Approved by:

Ghaith Rabadi (Director)

Resit Unal (Member)

Shannon Bowling (Member)

Murat Ermis (Member)

ABSTRACT

OPTIMIZATION MODELS AND APPROXIMATE ALGORITHMS FOR THE AERIAL REFUELING SCHEDULING AND RESCHEDULING PROBLEM

Sezgin Kaplan
Old Dominion University, 2011
Director: Ghaith Rabadi

The Aerial Refueling Scheduling Problem (ARSP) can be defined as determining the refueling completion times for fighter aircrafts (jobs) on multiple tankers (machines) to minimize the total weighted tardiness. ARSP can be modeled as a parallel machine scheduling with release times and due date-to-deadline window. ARSP assumes that the jobs have different release times, due dates, and due date-to-deadline windows between the refueling due date and a deadline to return without refueling. The Aerial Refueling Rescheduling Problem (ARRP), on the other hand, can be defined as updating the existing AR schedule after being disrupted by job related events including the arrival of new aircrafts, departure of an existing aircrafts, and changes in aircraft priorities. ARRP is formulated as a multiobjective optimization problem by minimizing the total weighted tardiness (schedule quality) and schedule instability. Both ARSP and ARRP are formulated as mixed integer programming models. The objective function in ARSP is a piecewise tardiness cost that takes into account due date-to-deadline windows and job priorities. Since ARSP is NP-hard, four approximate algorithms are proposed to obtain solutions in reasonable computational times, namely (1) apparent piecewise tardiness cost with release time rule (APTCR), (2) simulated annealing starting from random solution (SA_{random}), (3) SA improving the initial solution constructed by APTCR (SA_{APTCR}), and (4) Metaheuristic for Randomized Priority Search (MetaRaPS). Additionally, five regeneration and partial repair algorithms (MetaRE, BestINSERT, SEPRE, LSHIFT, and SHUFFLE) were developed for ARRP to update instantly the current schedule at the disruption time. The proposed heuristic algorithms are tested in terms of solution quality and CPU time through computational experiments with randomly generated data to represent AR operations and disruptions. Effectiveness of the scheduling and rescheduling algorithms are compared to optimal solutions for problems with up to 12

jobs and to each other for larger problems with up to 60 jobs. The results show that, APTCR is more likely to outperform SA_{random} especially when the problem size increases, although it has significantly worse performance than SA in terms of deviation from optimal solution for small size problems. Moreover CPU time performance of APTCR is significantly better than SA in both cases. MetaRaPS is more likely to outperform SA_{APTCR} in terms of average error from optimal solutions for both small and large size problems. Results for small size problems show that MetaRaPS algorithm is more robust compared to SA_{APTCR}. However, CPU time performance of SA is significantly better than MetaRaPS in both cases. ARRP experiments were conducted with various values of objective weighting factor for extended analysis. In the job arrival case, MetaRE and BestINSERT have significantly performed better than SEPRE in terms of average relative error for small size problems. In the case of job priority disruption, there is no significant difference between MetaRE, BestINSERT, and SHUFFLE algorithms. MetaRE has significantly performed better than LSHIFT to repair job departure disruptions and significantly superior to the BestINSERT algorithm in terms of both relative error and computational time for large size problems.

This dissertation is dedicated to *my beloved wife and my little Tigers.*

ACKNOWLEDGMENTS

It is a pleasure to convey my gratitude to people whose contribution in assorted ways to the research all in my humble acknowledgment.

In the first place, I heartily acknowledge Dr. Ghaith Rabadi, my advisor and dissertation chair, for his supervision, advice, guidance from the preliminary to the concluding level of this research. I would like to thank him for the long number of months writing and rewriting these chapters as well as giving me extraordinary experiences throughout the work. I am proud to record that I had several opportunities to work with an exceptionally experienced scientist like him. His guidance and professional style will remain with me as I continue my career.

I also thank my committee members, Dr. Resit Unal, Dr. Murat Ermis, and Dr. Shannon Bowling for their valuable recommendations pertaining to this study and assistance in my professional development. I gratefully thank them for their patience and guidance by constructive comments on my research in the midst of all their activity.

I gratefully acknowledge Dr. Ermis for his advice, and crucial contribution, which made him a backbone of this research and so to this dissertation. I am much indebted to him for his valuable advice in constructing the structure of the research, using his precious times to read this thesis from early stages and gave his critical comments about it.

I also would like to make a special reference to Dr. Resit Unal who taught how to use the valuable tools to complete this dissertation. His involvement with his originality has triggered and nourished my intellectual maturity that I will benefit from, for a long time to come. I would like to thank to Dr. Oktay Baysal and Dr. A. Osman Akan for their guidance in any respect during all stages of my Ph.D. education.

I am heartily thankful to my great country, Turkiye and its visionary commanders at the Turkish Air Force for giving this scientific research opportunity.

Words fail me to express my appreciation to my wife Seyma whose dedication, love, and persistent confidence in me, has taken the load off my shoulder. I owe her for being unselfishly let her intelligence, passions, and ambitions collide with mine.

I would like to thank my little tigers, Hakan and Neda, who gave me immeasurable moral support and encouragement. Your love is the greatest gift of all.

This thesis would not have been possible without my family. My wife's and my family deserve special mention for their all kind of help and support. It is a pleasure to express my gratitude to all of them.

Finally, I offer my regards and blessings to all of those who supported me in any respect during the completion of the dissertation.

NOMENCLATURE

ACO	Ant Colony Optimization
AFPAM	Air Force Pamphlet
AFRP	Aerial Fleet Refueling Problem
APTCR	Apparent Piecewise Tardiness Cost with Release Time
AR	Aerial Refueling
ARRP	Aerial Refueling Rescheduling Problem
ARSP	Aerial Refueling Scheduling Problem
ATC	Apparent Tardiness Cost
ATCS	Apparent Tardiness Cost with Setups
ATCSR	Apparent Tardiness Cost with Setups and Ready Times
ATP	Allied Tactical Publication
AWACS	Airborne Warning and Control Systems
BDA	Boom Drogue Adapter
CL	Candidate List
COMSOAL	Computer Method of Sequencing Operations for Assembly Lines
COVERT	Cost Over Time
d-to-D	Due date-to-Deadline
DM	Decision Makers
DOE	Design of Experiment
EDD	Earliest Due Date
FIFO	First-In-First-Out
GA	Genetic Algorithms
GAO	General Accounting Office
GRASP	Greedy randomized Adaptive Search Procedure
GTTS	Group Theoretic Tabu Search
LP	Linear Programming
LPT	Longest Processing Time
LSHIFT	Left Shift
MetaRaPS	Metaheuristic for Randomized Priority Search
METARE	Rescheduling with MetaRaPS algorithm

MIGP	Mixed-Integer Goal Programming
MILP	Mixed Integer Linear Programming
MOD	Modified Operation Due date
MOO	Multi Objective Optimization
MSLACK	Minimum SLACK
MTWT	Minimum Total Weighted Tardiness
NATO	North Atlantic Treaty Organization
NP	Non-Polynomial
NSGA	Neighborhood Search in Genetic Algorithm
NSIAD	National Security and International Affairs Division
OPL	Optimization Programming Language
RM	Rachamadugu and Morton
SA	Simulated Annealing
SEPRE	Sequence Preserving Rescheduling
S/N	Signal-to-Noise
SOO	Single Objective Optimization
SPT	Shortest Processing Time
S/RPT	Slack per Remaining Processing Time
TS	Tabu Search
TWT	Total Weighted Tardiness
UAV	Unmanned Aerial Vehicle
US	United States
VNS	Variable Neighborhood Search
WLPT	Weighted Longest Processing Time
WSPT	Weighted Shortest Processing Time
X-RM	Modified Rachamadugu and Morton

TABLE OF CONTENTS

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

Aerial refueling (AR), also called air refueling, in-flight refueling, air-to-air refueling, is the process of transferring fuel from one aircraft (a tanker) to another (a receiver) during flight. AR is extensively used in large-scale military operations because of its advantages for an air force in terms of responsiveness, endurance, flexibility, efficiency, and safety. Aerial refueling is generally employed in two cases of military operations: inter-theater and in-theater. Inter-theater operations contain the deployment of forces (e.g., overseas deployments) and its accompanied air refueling requirement. The second role of aerial refueling is to support in-theater operations. This support includes deployments and employments during a conventional conflict (e.g., Operation Desert Storm of the U.S. Air Force) when many strike aircrafts are conducting attacks around the clock and spread across a local area.

Scheduling as a decision making process that deals with allocation of resources to tasks over given time periods is needed for effective aerial refueling while maintaining safety and efficiency. The Aerial Refueling Scheduling Problem (ARSP) can be defined as determining the assignment of each fighter aircraft (job) to tanker (machine) and the refueling completion times for the aircrafts. Best of our knowledge, there is very little existing research on aerial refueling scheduling problems, although it has significant effects and advantages for air operations.

This dissertation research examines the in-theater ARSP which has further research paucity than inter-theater ARSP. The in-theater ARSP holds some different characteristics from the inter-theater ARSP. It is far more complex and difficult to support because of shorter planning periods, rapidly changing priorities, crowded airspace, less predictable fuel requirements, lack of standardized refueling equipment, and continuous operations by a lot of aircrafts. Therefore, dealing with in-theater environment of aerial refueling not only has crucial effects to air force dominance, but also valuable intellectual contributions to scheduling theory applications.

In-theater AR is supported entirely through AR tracks which are similar to gas stations floating in the sky. Fighter wings are even-numbered groups of fighters that get refueled by tankers in the air. Tankers orbit in a track location with a constant

speed and altitude waiting for receivers to arrive for refueling. Receivers may be in different types of bombers, airborne warning and control aircrafts, cargo aircrafts, fighters, helicopters, and also other tanker aircrafts. Tankers may also have different types in size, weight, take off fuel load capacity, offload (deliverable fuel) capacity, maneuver capability, fuel transfer rate, and refueling systems.

ARSP has some certain problem assumptions to scope this dissertation research. First, fighter aircrafts are considered AR receivers and tankers are identical aircrafts with only one type of refueling system that is compatible with typical fighters used by many air forces. Multiple tankers are required to shorten the refueling time and keep fighters on schedule in crucial time periods. It is assumed that track stations for the tankers are known, the nnumber of tankers does not change during an operation. For the purposes of this research, tankers are assumed to be continuously available. In order to maintain fuel continuity, just before a tanker completes its mission (i.e. just before running out of fuel), another tanker will replace it. Second, a job of the ARSP can be defined as AR process of a wing. Fighters execute their missions staying together as a group of an even number, generally four identical fighters. Third, refueling requirements call for wings to refuel at different times. Moreover, it is assumed that fighter wings which move dynamically in the sky, can reach to available tankers in equal times. Communication between a tanker and receivers in the air occur perfectly; in other words, data can be sent accurately between the tanker and the receivers without delays. Fourth, there is no setup requirement between refueling wings (jobs) due to compatible and unique refueling systems. Approaching and anchoring of the wing can be considered a setup time but this setup time is not sequence-dependent; therefore, it can be considered as a part of the processing time.

Beyond these assumptions, ARSP has some characteristics that help mapping in scheduling theory. First, higher availability and less tardiness of the wings are desired in order to hold air force dominance during air operations. Besides, AR priorities affect the scheduling. Therefore, a tardiness penalty, which takes into account wing priorities, was defined as a scheduling performance measure. Second, a time constraint associated with fuel consumption must be defined because there is a maximum waiting time for each wing to refuel before it reaches its minimum fuel level; otherwise a wing has to return back to base. Returning back without refueling causes a large penalty due to operational availability issues.

On the other hand, resources commonly occur in parallel and many real life problems can be modeled as parallel machine scheduling problems. In the classical parallel machine scheduling problem, there are n jobs and m machines. Each job needs to be executed on one of the machines during a fixed processing time. A parallel machine scheduling problem involves both resource allocation and sequencing. It allocates jobs to machines and determines the sequence of jobs on allocated machine. Machines may be identical where the processing time of each job is independent of the assigned machine, uniform where each machine may have a different speed, or unrelated where the processing time of each job is dependent on the assigned machine without a particular relationship. The aim is to find the schedule optimizing a certain performance measure such as makespan, maximum lateness or weighted tardiness.

In this dissertation, the research was extended to address the rescheduling problem to have a more realistic and robust scheduling system. Despite the extensive literature on scheduling problems, little research exists on rescheduling parallel machines. One of the difficulties is sourced from the dynamic environment of the AR process where disruptions caused by dynamic and unexpected events are common. AR rescheduling problem (ARRP) is defined as updating the existing AR schedule after being disrupted by a job related event. Although more disruptions may be considered for ARRP, job related disruptive events are studied in this dissertation since they occur much more frequent than other type of disruptions in air operations. Events that have potential to cause significant disruptions in the AR schedules are interpreted by the arrival of new jobs, departure of an existing job, and changes in job priorities characterized by a combined change of weight and due date. In the ARRP, continuous rescheduling approach in which updating the existing schedule takes place when an event occurs is performed. The objective in the ARRP will not only be maximizing the schedule performance, but also minimizing schedule instability to decrease contact requirements under tight communication constraints and to avoid excessive rescheduling computation time.

1.1 Aerial Refueling

By the end of the Vietnam War, refueling on the tanker without landing became a routine part of combat air operations for fighters and bombers as a result of its importance for an air force. It is extensively used in air operations, because it has some advantages for an air force seeking dominance against enemy forces.

1. Responsiveness: Air refueling provides rapid response by remaining airborne more and reducing time to get ready. It allows fighter to deploy and strike targets deep in enemy territory.

2. Endurance: It greatly extends the operational range of air forces and the time that fighter aircraft can protect friendly forces from attack by enemy aircraft.

3. Flexibility: By reducing the logistical trail needed for an air operation, it enables taking off with a larger payload which could be weapons, cargo or personnel, opening up new operational capabilities.

4. Efficiency: Non-stop flying capabilities give each aircraft the ability to reach its destination before its required arrival time.

5. Safety: Combat aircrafts can avoid reliance on other countries for land-based refueling stations so that they can be based in safe areas.

The inter-theater AR, one type of aerial refueling, emphasizes planned missions in which each tanker provides a large and predictable amount of fuel over great distances. This type of AR utilizes tankers to deploy receivers along a pre-prepared route. A considerable portion of the route is flown often with the receiver and escorting tanker in company over large bodies of water (ATP-3.3.4.2 (ATP-56(B) Change 1), 2008). The important point for the inter-theater AR is that tankers need to be assigned to receiver groups and travel for them.

On the other hand, the in-theater AR is far more complex and difficult to support because of shorter planning periods, rapidly changing priorities, crowded airspace, less predictable fuel requirements, lack of standardized refueling equipment, and continuous operations by a lot of aircrafts (GAO/NSIAD-94-68, 1993). In-theater AR procedures have following characteristics unlike inter-theater AR although they have similar AR processes.

Refueling Tracks: In-theater AR is supported entirely through AR tracks (e.g., tracks 1 and 2 in Figure 1), which are similar to gas stations floating in the sky. Tankers, which are considered high valued assets for an air force, orbit in a track location with a constant speed and altitude waiting for receivers to arrive for refueling. Track stations are decided on the ground among the alternative stations according to some criteria such as operational effectiveness, flight safety under weather conditions and enemy attacks, communication efficiency between tanker and receivers, and international agreements. Moreover, the anchor rendezvous tracking that is shown in the Figure 2 with its track parameters is the simplest one among different types of refueling rendezvous tracking.

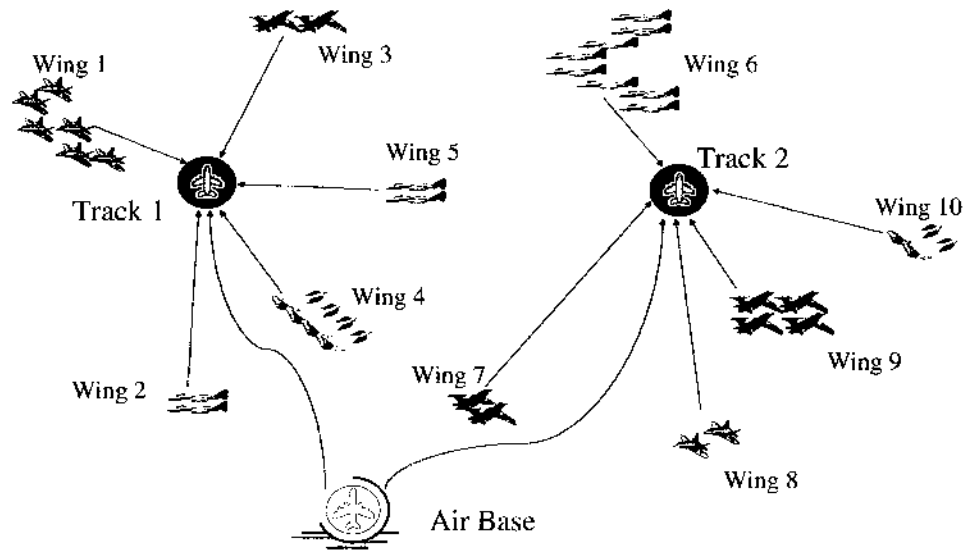


Figure 1 In-Theater Aerial Refueling

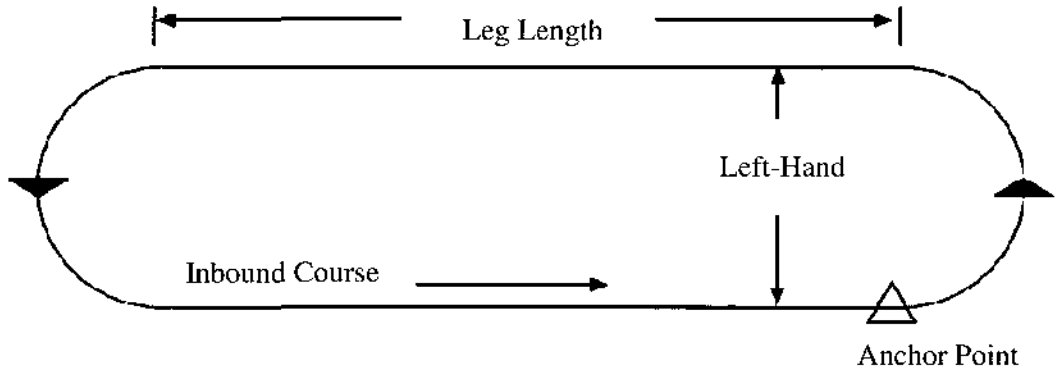


Figure 2 Anchor Rendezvous Tracking

Receivers and Tankers: Receivers may be different, such as large and heavy bombers, Airborne Warning and Control Systems (AWACS), cargo aircrafts, also other tanker aircrafts or fighters and helicopters. However, AR receivers are often *fighter* aircrafts. Tankers may also have different types in terms of size, weight, take-off fuel capacity, offload capacity, maneuver capability, fuel transfer rate (ATP-3.3.4.2 (ATP-56(B) Change 1), 2008). For example, the current tanker fleets of NATO countries mostly consist of KC-135 and KC-10 jets. The KC-135 typically carries 180,000 lbs. of fuel while the larger KC-10 holds 327,000 lbs. according to the standard planning factors. The track parameters in Figure 2 are also important for tanker planning because if tankers fly a mission with radius of 500 nm (nautical miles) then the KC-10 would have 233,500 lbs. of fuel to offload (deliverable fuel after subtracting tanker's fuel consumption) while the KC-135 has 122,200 lbs (AFPAM 10-1403, 2003).

Refueling Systems: There are two different AR systems in use: Probe/Drogue and Boom. As shown in Figure 3, a tanker trails two or more hoses in the *probe and drogue* system while the tanker is fitted with one flyable, telescopic boom in the *boom* system. Although these two systems are not compatible, some booms can be adapted (on the ground) using a boom drogue adapter (BDA) kit. This makes the boom compatible with probe equipped receivers. Some tankers (e.g., the KC-10A) are equipped with both boom and hose/drogue systems and either may be used on the same flight (ATP-3.3.4.2 (ATP-56(B) Change 1), 2008). A single flying boom can

transfer fuel at approximately 6,000 lbs. per minute. A single hose-and-drogue can transfer 1,500-2,000 lbs of fuel per minute.

AR with Probe and Drogue AR with Boom

Figure 3 Aerial Refueling System Types

However, fighters cannot accept fuel at the boom's maximum rate. Typical fighters can accept fuel at 1,000 to 3,000 lbs. per minute either from the boom or the hose-and-drogue (Bolkcom, 2006).

1.2 Mapping the ARSP to Abstract Scheduling

Air force responsiveness and endurance require joining back to operation mission as soon as possible after breaking the active mission for refueling. Thus, scheduling as a decision making process is needed for effective AR while maintaining safety and efficiency in the air operations. The AR Scheduling Problem (ARSP) can be defined as determining the assignment of each fighter aircraft (job) to tanker (machine) and the refueling completion times for the aircrafts. To map ARSP to an abstract and classical scheduling problem, we consider the following general and frequently used scheduling assumptions:

1. Any job can be processed on at most one machine at any time.
2. No preemption is allowed where once an operation is started it is continued until complete.

3. Ready times of all jobs are zero, i.e. all jobs are available at the commencement of processing.

4. Machines are always available and reliable (i.e., rarely breakdown).

5. Each machine can process at most one job at any time.

6. Setup times are sequence-independent and are included in processing times.

7. Processing times, technological constraints, weights, and due dates are deterministic and known in advance where appropriate (Blazewicz et al., 2007).

For the ARSP, the following scheduling assumptions are considered:

Machine Related Assumptions: Equipment, performance, and procedural compatibilities between tankers and fighters are important for a successful AR. So, it is assumed in this dissertation research that AR receivers are *fighter* aircrafts and tankers in the refueling missions are *identical* KC-135 type aircrafts with only the boom type which is compatible with typical fighters used by many air forces. Thus, tankers tracking on different locations can be considered *identical parallel machines* from the scheduling theory point of view. *Multiple tankers* are needed to shorten the refueling time and keep fighters on schedule because many fighters can be over target areas at the same time. Multiple tanker tracks are placed to maintain high volume in crucial time periods. It is assumed that no new tanker stations are added and the number of the multiple tanker tracks is fixed. Moreover, continuous fuel supply and tanker availability is assumed in this research. Fuel stock considerations and crew constraints require specifying mission duration for tankers. The flight schedule is limited by the 12-hour operational day and must allow for transit time to and from theater, one-hour pre-flight preparation, two-hour post-flight maintenance service, and at least a three hour turnaround time between refueling missions. However, for continuous fuel supply, just enough time before finishing one tanker's mission, another tanker will replace it with no interruption. After taking over, the preceding tanker returns back to the main base. Also, it is assumed that tankers do not refuel each other.

Job Related Assumptions: A job represents a wing in ARSP. Fighters execute their missions in the form of a wing which is composed of an even number of identical fighters, as shown in Figure 1, because of flight safety and compatibility issues. Therefore, fighters in a wing approach the tanker together and the refueling process is carried out one fighter at a time while the rest of the wing waits in the observation area or reform area as shown in Figure 4. Upon completion, the wing leaves the tanker as a group. Furthermore, a job refueling process *cannot be interrupted* so as not to decompose the unity of the wing. Wings are assumed independent in the sense that there is no precedence dependence between them which fits the flexibility necessary for an air operation.

Ready-Time Related Assumptions: Ready time can be defined as the earliest contact time of a wing with a tanker to request refueling when its fuel level is reduced to a safe fuel level. In other words, it is the earliest time that a wing is available and can leave from its mission area to refuel. Fuel burning rate depends on the mission type with different maneuver requirements, mission locations at different altitudes, weather conditions, deployment distances and fighter characteristics such as fuel capacity, speed, weight, and pilot skills.

Consequently, refueling calls of the wings to tanker will be at different times. This means that all wings are not available to refuel at the same time and they have *different ready times*. Moreover, it is assumed that variability in distances between individual wings and each tanker are negligible. Consequently wings can arrive at different tankers with equal times. Data are sent between the tanker and the receivers without delays and errors and communication between the tanker and receivers in the air is assumed perfect. However, a contact between the tanker commander and the wing leader must be established at least 15 minutes prior to the refueling rendezvous time for approaching to waiting position.

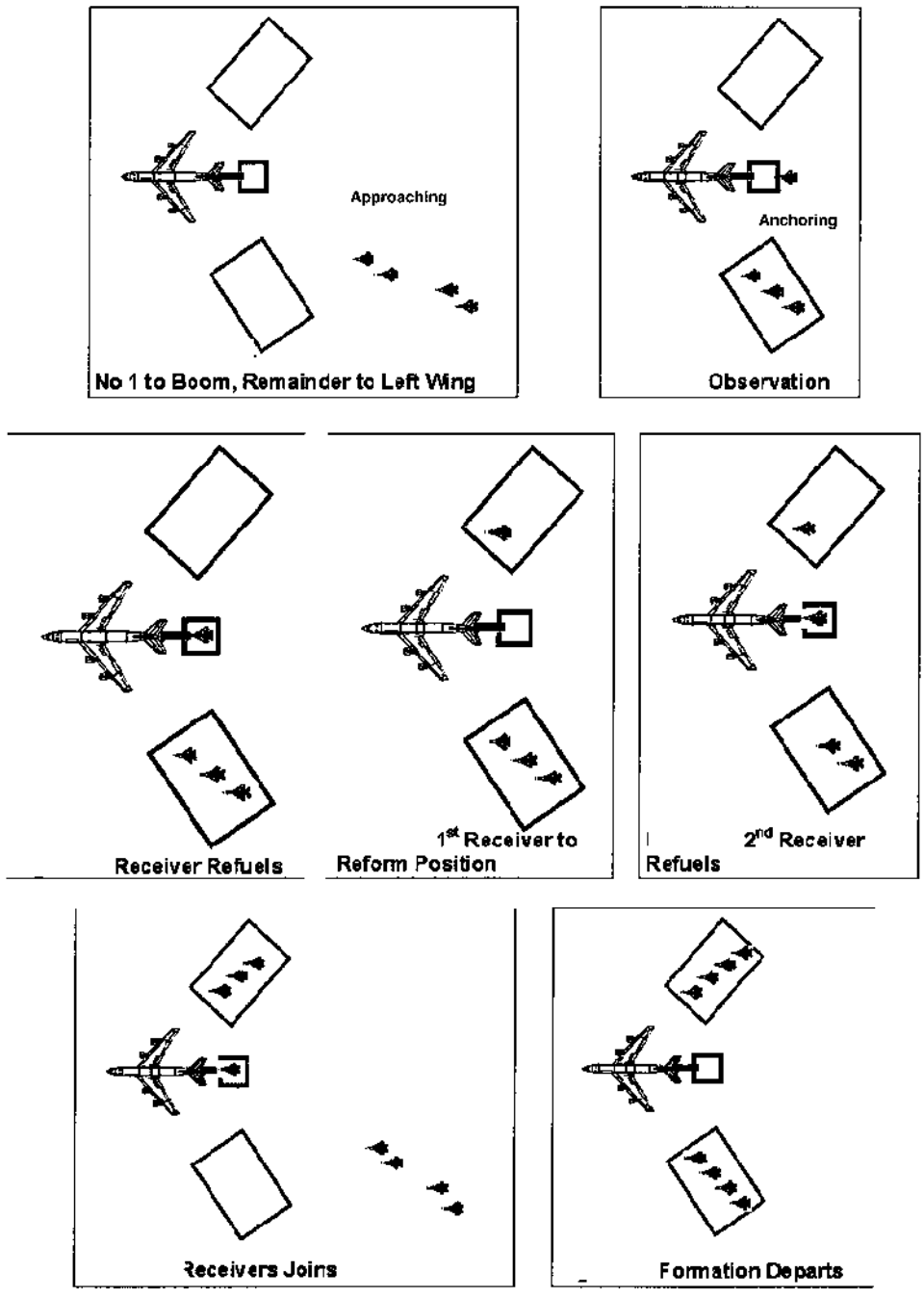


Figure 4 Wing Refueling (ATP-3.3.4.2 (ATP-56(B) Change 1), 2008)

Setup Related Assumptions: There are no setup requirements between the wing AR jobs as a result of the boom system. The processes in Figure 4, such as approaching and anchoring, can be considered as setup time. But setup time is *not sequence dependent* and it can be included in processing time. Thus, aggregate processing times of the jobs include time required to approach the tanker, connecting (anchoring) and disconnecting times of each receiver, and refueling times of each receiver.

Beyond its assumptions, ARSP has some characteristics that can also be mapped to classical scheduling. First, *weighted tardiness* can be used as a tardiness penalty to evaluate the due date delivery performance of schedules. It is extremely important to maintain higher availability and less tardiness of wings in order to hold air force dominance during air operations. Thus, there is a latest time for a wing to join the mission and the tardiness penalty is essential for the scheduling problem. Furthermore, operation conditions, capabilities of different types of fighters, experience, and skill levels of the wings and assessments of mission planners affect the importance of a wing and also its AR priority affects the scheduling. So the tardiness penalty has to take into account the priorities. Second, a *time (d-to-D) window* constraint associated with fuel consumption is required. *Bingo level* represents the required minimum fuel level at which the wing must return back to base where the fuel amount would barely be enough to return. Therefore, there is a maximum waiting time to start refueling for a wing otherwise it has to return back to base. Returning back to base without refueling (i.e., not assigning a job to a machine) causes a large penalty due to operational dominance issues.

Finally, for both ground and air optimal scheduling decisions, the basic questions ARSP are the following:

- Which wings (jobs) should be assigned to which tanker (machine)?
- Which wings could not assigned and must return to base?
- What time does each wing start to refuel?
- What is the optimal total cost for a given schedule given the objective function defined?

1.3 Problem Statement

ARSP can be defined as scheduling n jobs (wings of aircrafts) on m identical parallel machines (tankers) where job j arrives (becomes available) at ready time r_j and should be complete by the due date d_j and before deadline D_j. The problem elements are shown in Figure 5. The following notation will be used throughout this dissertation:

Processing time (p_j): Processing time of the receiver or wing (job) j is the sum of approaching time to refueling area, anchoring times to tanker and the fuel pumping times of all fighters in wing j.

Ready time (r_j): The earliest time a receiver j can start processing (i.e., first contact with a tanker to request refueling when its fuel level reduced to safe fuel level). It is the earliest time that a receiver is able to leave from mission area to refuel. Thus, a receiver j cannot be scheduled before r_j.

Weight (w_j): The importance of wing j for the air operation. It is determined according to operation conditions, capabilities of fighters, experience, and skill levels of the wings as well as the assessments of mission planners.

Bingo level time (b_j): The latest time before which wing j has to be scheduled; otherwise it will go back to base due to low levels of fuel.

Due date (d_j): Planned latest date of receiver j to complete refueling. Completion of the refueling job after the due date is allowed, but then a weighted tardiness penalty is incurred.

Deadline (D_j): The latest date of a receiver to finish refueling after which it must return to base to avoid running too low on fuel level. It is the upper bound of the d-to-D window. It can be calculated simply by adding the processing time to bingo level time.

Start time (s_j): Starting time of receiver j to refuel.

Completion time (Cj): Refueling process completion time ($s_j + p_j$) of receiver j.

Waiting time ($s_j - r_j$): The time between refueling request and the refueling start time. During this time, wings continue their missions.

Due date-to-Deadline window (D_j -d_j): The time window in which a weighted tardiness cost is incurred. Missing the due date is not preferred but allowed and a weighted tardiness cost will be incurred for jobs that miss their due date but not their deadline. If a job misses D_j, it must return to base incurring a high penalty and will not be assigned to a machine. Note that there is also a scheduling window between ready times and deadlines in which all jobs have to be started and completed.

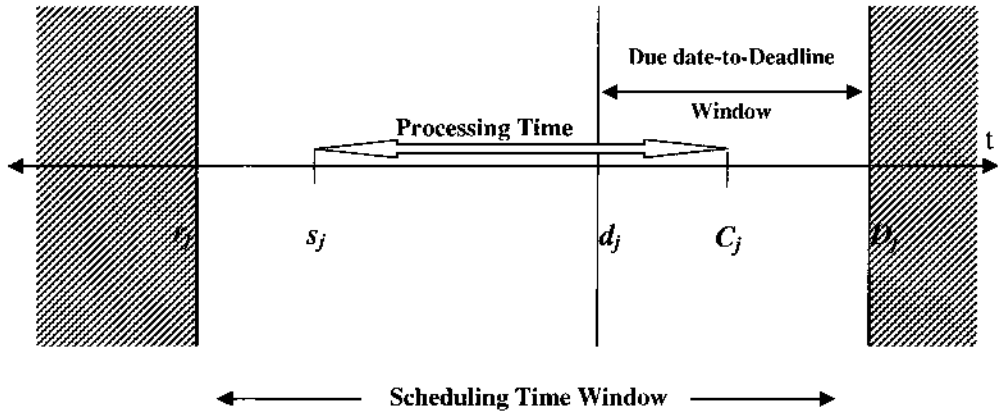


Figure 5 Scheduling Time Horizon

A piecewise tardiness cost function may be defined for the problem where if $D_j > C_j > d_j$ then the tardiness cost $T_j = \max (C_j-d_j, 0)$; while if $C_j > D_j$ job j is not scheduled and a high *unavailability cost, F* will be incurred, as shown in Figure 6. One can think of F as the cost of wing's failure to refuel and returning to base without completing its mission. The job is labeled *unavailable* and *not scheduled,* if its completion time misses the deadline.

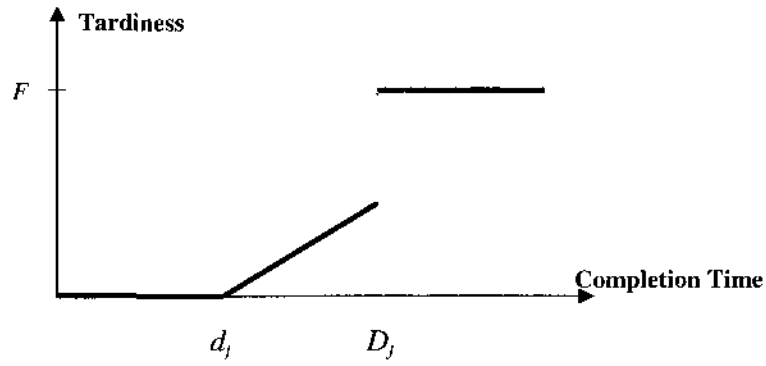


Figure 6 Piecewise Tardiness

A scheduling problem is described by a triplet $\alpha \mid \beta \mid \gamma$. The α field describes the machine environment; the β field provides details of processing characteristics and constraints. Finally, the γ field describes the objective to be minimized and often contains a single entry (Pinedo, 2008). Resources such as production lines for products, docks for ships, teachers for student groups, hospital assistance for patients, etc. commonly occur in parallel and many real life problems can be modeled as parallel machine scheduling. In the classical parallel machine scheduling problem, there are n jobs and m machines. Each job needs to be executed on one of the machines during a fixed processing time (Mokotoff, 2001). Parallel machine scheduling problems involve both resource allocation and sequencing. It determines which jobs have to be allocated to which machines and determines the sequence of the jobs allocated to each machine. Machines can be *identical* that the processing time of each job is independent of the assigned machine, *uniform* that each machine has a different speed, or *unrelated* where the processing time of each job is dependent on the assigned machine without a particular relationship. The aim is to find the schedule that optimizes a certain performance measure such as makespan, maximum lateness, or weighted tardiness. The minimization of the total weighted tardiness is important in scheduling since maintaining quality of service usually has an objective with weights (Pinedo, 2008).

In the ARSP, it is assumed that ready times of all wings to identical parallel tankers are different with d-to-D window constraint to minimize total weighted tardiness. ARSP can be defined as scheduling n jobs (wings of aircrafts) on m identical parallel machines (tankers) where job j arrives (becomes available) at ready time r_j and should be complete by the due date d_j and before deadline D_j. Therefore, the ARSP is denoted by $P_m|r_j, d\text{-}to\text{-}D\ window|\Sigma w_j T_j$.

In order to illustrate the problem, sample data (not real life data) are given in Table 1.

j	r_j	p_j	w_j	d_j	D_j
1	0	30	1	60	100
2	10	20	3	70	100
3	20	40	5	80	130
4	20	20	1	80	110
5	30	30	3	90	130
6	40	40	5	100	150
7	50	30	1	110	150
8	50	20	3	110	140
9	60	40	5	120	170
10	70	20	1	130	160

Table 1 Sample Data for Pm|rj, d-to-D window|ΣwjTj

In the sample problem, there are two identical parallel machines (1 and 2), ten jobs (1-10), three types of jobs (20, 30 and 40 units of processing time), three priority levels (1, 3 and 5), and fixed 200 unit unavailability cost (f) if a job j is to miss D_j. A feasible schedule (shown in Figure 7) is provided by Weighted Longest Processing Time first (WLPT) rule; one of the most widely used dispatching rule (greedy heuristic).

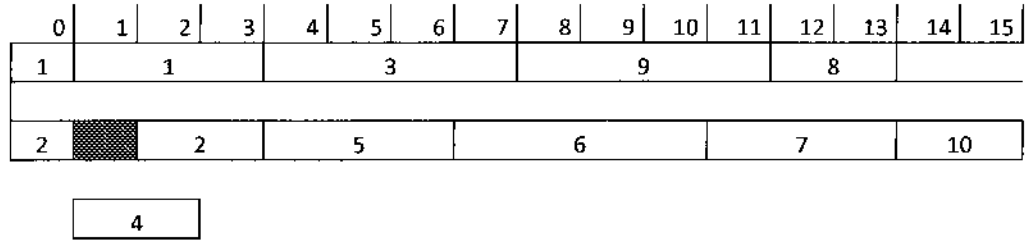


Figure 7 A Feasible Schedule

Job (j)	Ready Time (r_j)	Processing Time (p_j)	Priority (w_j)	Due Date (d_j)	Deadline (D_j)	Machine (M_j)	Schedule Time (s_j)	Waiting time (s_j-r_j)	Completion Time (C_j)	C_j-d_j	Weighted Tardiness	Return Back (0/1)
1	0	30	1	60	100	1	0	0	30	-30	0	0
2	10	20	3	70	100	2	10	0	30	-40	0	0
3	20	40	5	80	130	1	30	10	70	-10	0	0
4	20	20	1	80	110	1	130	110	150	200	200	1
5	30	30	3	90	130	2	30	0	60	-30	0	0
6	40	40	5	100	150	2	60	20	100	0	0	0
7	50	30	1	110	150	2	100	50	130	20	20	0
8	50	20	3	110	140	1	110	60	130	20	60	0
9	60	40	5	120	170	1	70	10	110	-10	0	0
10	70	20	1	130	160	2	130	60	150	20	20	0

Table 2 Scheduling Results for Sample Data

The results are given in Table 2. Binary unavailability variable (return back column) takes 1 value if the job missed the deadline; otherwise it takes 0. For this case, the schedule has a total weighted tardiness of 300 with 1 unprocessed (return to base) job.

1.4 Difficulties in the ARSP

In the planning stage before tankers and wings take off, planners try to calculate the refueling request times and tanker schedules. All risks and contingencies are to be analyzed and the provided arguments are to be exploited to create proactive schedules on the ground. On the other hand, the tanker crews gather information from all wings, plan the new refueling schedule, and send the results back to the wings in the air. Therefore, there are some difficulties that make ARSP different from an ordinary parallel machine scheduling problem. These difficulties have emerged as a result of dynamic, uncertain, multi objective nature of the problem, unexpected emergent events, variability of its elements, and constraints.

1. *Dynamic Environment:* Set of wings in the air, wing priorities and ready times change in real life.

2. Set of jobs in the theater may change during the operation. New rush job arrivals or leavings occur according to operational decisions.

3. It is known that fighters are configured according to some special fighting functions and capabilities. Meanwhile, the course of the air operation and the new target priorities require different fighting functions. Therefore, needs for each differently configured wing and given job priorities may change according to air operation conditions.

4. Planned ready times may be shifted backward or forward by more than a little deviation according to revised mission plans due to unexpected attacks and future operation missions.

Uncertain Environment: Processing time, ready time, due date, deadline, and priority are uncertain.

1. Calculation of the processing times is hard work because it comprises the approaching time, the boom engagement/disengagement times and fuel pumping time of all fighters in the wing. Eventually, the elements of processing time may deviate due to weather conditions, pilot skills, and inefficiency of refueling system or night conditions.

2. Planned ready times may deviate according to changes in fuel level burning rate.

3. Similarly, due dates and deadlines cannot be taken as exact times.

4. Factors determining job priorities such as current situation assessments of decision makers, scaling urgency, and the relationship between operation conditions and fighter functions have a subjectivity and fuzziness nature.

Multi Objectives: Besides minimization of total weighted tardiness, other objectives related to dumped offload fuel of tanker and waiting time of wings may be accounted for in the problem. The ability of air refueling operations to support combat missions is limited not only by the tardiness of tankers but also by the efficiency of fuel transfer. Utilization of the tanker in terms of unused fuel is an indication of the inefficiency of tanker operations (GAO/NSIAD-94-68, 1993). For example, tankers often returned to base with a large amount of unused fuel (on average, 40 percent of the fuel carried by tankers), some of which has to be dumped in order for the tanker to land. The cost of the inefficiency, in other words, total idle times of each tankers

should be minimized in scheduling. Moreover, in order to reduce the amount of fuel burned by wings, total waiting time of the jobs should be minimized.

Emergency Cases (Tanker Availability): An emergency in either the tanker or receiver may cause an urgent stop during AR, if the following are the case:

1. There is a sudden attack to the tanker. The tanker has to move to another location.

2. There is an unexpected fuel shortage of the tanker. It has to return back to base. Due to continuous tanker availability, if an unplanned condition is recognized, the next ordered tanker can be called earlier as a prevention mechanism.

3. The receiver is judged to be flying erratically by the tanker. A faulty receiver may be warned or not be allowed to refuel. This situation does not affect the tankers' availability.

4. The tanker has a malfunction. In actuality, boom failures typically occur when receivers accidentally dislodge the drogue during the refueling process. However, many boom failures do not last the duration of the tanker on-station.

5. Bad weather conditions do not allow refueling. Refueling may have to be cancelled anyway or moved to another location.

Most emergency cases are associated with *tanker availability*. However, it is fact that these cases very rarely occur in real life. Besides unexpected tanker availability related events, some conditions about breakdown, maintenance, weather, etc. for each tanker may be known or forecast on the ground. If the continuous tanker supply assumption is violated, these tanker availability constraints should be accounted for in scheduling. Moreover, *flexible set of tankers* is possible when the number of unscheduled wings reaches an unallowable level as a result of high volume air operation.

Processing time may be defined as dependent on the scheduled refueling starting time which is a decision variable. Processing times inherently depend on the quantity of fuel delivered. If the waiting time lengthens, the fighter will engage to refuel at lower fuel levels and consequently more processing time will be required.

Different types of tankers, such as KC-10 tanker aircrafts and tankers with PDA Kits may be included in the model as a resource. In this case, some compatibility constraints come out and a particular wing may be refueled by a subset of machines. On the other hand, speed differences will not matter because refueling speed is determined by the receiver capability.

The previous wing form can be divided into subgroups of fighters before refueling to increase the utilization of tankers. Subgroups can also bring flexibility to cope with fuel burning rate deviations caused by different maneuvers and pilot skills unless deviations cannot be compensated by sequencing rearrangements within the wing. However, because of unity desire, a unity penalty or simultaneous refueling constraint may be used for dividing the wing into subgroups.

It is essential that an efficient coordination and communication interface exists between tanker and receiver to ensure the correct positioning and timing of the tanker to meet receiver demands (ATP-3.3.4.2 (ATP-56(B) Change 1), 2008). Nevertheless, all tankers and receivers may have to stand out against communication restrictions during operations.

1.5 Research Scope and Objectives

This dissertation research examines the aerial refueling scheduling problem (ARSP) and the aerial refueling rescheduling problem (ARRP) for *in-theater* operation environment. The research scope is framed around two areas: parallel machine scheduling and rescheduling. The scheduling/rescheduling environment has a finite set of jobs and a finite set of machines. ARSP assumes that the jobs have different release times and a due date-to-deadline (d-to-D) window between the refueling due date and deadline before which an aircraft wing (job) will to return to base without refueling. Minimizing the total weighted tardiness is a good objective function to meet the aircrafts' refueling due dates and ultimately the mission's due date. Consequently, a piecewise tardiness cost was defined by taking into account d-to-D windows and job priorities to evaluate the schedule's quality. Thus, ARSP can be modeled as a parallel machine scheduling with release times and d-to-D window to minimize total weighted tardiness.

In the ARRP, continuous rescheduling approach is used where updating the existing schedule takes place only when an event occurs. That is, it is assumed that schedules will be updated only as a result of job related disruptions such as the arrival of new jobs, the departure of an existing job, and changes in job priorities characterized by a combined change of weight and due date. ARRP considers both minimizing total weighted tardiness as a primary measure of schedule performance, as well as minimizing schedule instability as a measure of disruption caused by the rescheduling. Instability of the AR schedules is defined here as any changes in job starting times on the assigned machine. Then the measure of schedule instability can be defined as the proportion of rescheduled jobs that change machine assignment and/or starting time. In the context of air operations, a communication event with the pilots to either change their start time or their tanker assignment is considered as an event that will change the original schedule and therefore there is no distinction between them from rescheduling pespective. Consideration of the two objectives of schedule quality and stability leads to the formulation of ARRP as a multi objective scheduling problem.

The objectives of the research can be summarized as follows:

1. To formulate a scheduling problem from the military environment as an abstract parallel machine scheduling problem that is widely used in production environments. The details of the formulation for some aspects of the problem are unique to the problem addressed in this research.

2. To find optimal solutions for the problem using mixed integer programming

3. To develop an effective and efficient solution methods to obtain best initial schedules that satisfy the constraints of the scheduling problem in a reasonable time. This is accomplished by introducing a dispatching rule for the problem.

4. To find near-optimal solutions using metaheuristics including Simulated Annealing and Meta-RaPS.

5. To address the rescheduling problem by formulating the problem as a a multi objective optimization problem of minizing both the weighted tardiness and instability of the schedule. This is accomplished using optimization models and approximate algorithms to update the disrupted schedules satisfactorily

with respect to both objectives, and to effectively evaluate the trade-off between the objectives.

The rest of this dissertation is organized as follows. In Chapter 2, related research is summarized. Optimization models and approximate algorithms developed for the ARSP problem are introduced in Chapter 3including a mixed integer linear programming model (MILP) for optimal solutions and metaheuristic algorithms to find near optimal solutions. Computational studies for ARSP are presented and their results are analyzed in Chapter 4.Rescheduling mechanisms and solution methods for ARRP are developed in Chapter 5 followed by a computational study for small and large problem sizes in Chapter 6. Finally, conclusions and future research are presented in Chapter 7.

CHAPTER 2

LITERATURE REVIEW

In light of above issues, related literature including ARSP applications, parallel machine scheduling with unequal release times, and d-to-D window constraints to minimize total weighted tardiness and rescheduling, were reviewed to display practical and intellectual contributions of the dissertation research.

2.1 Aerial Refueling Scheduling Problem (ARSP)

The *only* existing research on scheduling in-theater AR is Jin et al. (2006). They introduced the static autonomous refueling scheduling of multiple unmanned aerial vehicles (UAVs) as a combinatorial optimization problem. In the problem, a *single* tanker needs to provide refueling service for multiple UAVs. The number of UAVs is known and does not change during the refueling process. Each UAV has different parameters such as current fuel level, refueling time, and return-to-field priority that are effective for scheduling. A dynamic programming method was used to develop an efficient recursive algorithm to find the optimal initial sequence for the AAR scheduling problem. Furthermore, they considered the optimal sequence recalculation necessity due to conditions change, such as joining or leaving the queue of UAVs unexpectedly. They developed a systematic shuffle scheme to reconfigure the UAV sequence using the least amount of shuffle steps. They proposed a rearrange scheme in which only one UAV will be shuffled at each time and the cost for each movement is identical. Since the movements consume fuel/time and introduce disturbances, they considered them as a reconfiguration effort and integrated into the dynamic programming algorithm, when searching for the new optimal sequence.

On the other hand, the only major work on inter-theater AR, Barnes et al. (2004), studied the aerial fleet refueling problem (AFRP) and used a Group Theoretic Tabu Search (GTTS) approach for solution.

2.2 Scheduling Identical Parallel Machines with Ready Times to Minimize Total Weighted Tardiness Problem

There is very little existing research, as shown in Table 3, addressing the $P_m|r_j|\Sigma w_jT_j$ problem. However, the related research does not consider time windows or other

difficulties defined for ARSP. The most closely related works are Mönch et al. (2005), Reichelt et al. (2006), Pfund et al. (2008), Gharehgozli et al. (2009), and Driessel and Mönch (2009). They differentiate sequence dependent setup, batch machines, and precedence constraints. Recent researches are concentrated on sequence dependent setup constraints. Only Reichelt et al. (2006) and Gharehgozli et al. (2009) introduced multi objectives. Generally, dispatching rules, exact algorithms, heuristics based on dispatching rules, local searches, different types of metaheuristics such as genetic algorithms, variable neighborhood searches, and hybrid metaheuristic combined by NSGA-II are used for this type of problem.

Mönch et al. (2005) attempted to minimize total weighted tardiness on parallel batch machines with incompatible job families and unequal ready times of the jobs ($P_m|r_j$,batch,incomp.$|\Sigma w_j T_j$). Given that the problem is NP-hard, they proposed two different decomposition approaches. The first approach forms fixed batches then assigns the batches to the machines using a genetic algorithm (GA) and finally sequences the batches on individual machines. The second approach first assigns jobs to machines using a GA then forms batches on each machine for the jobs assigned to it and finally sequences the batches. Dispatching and scheduling rules were used for the batching phase and the sequencing phase of the two approaches. In addition, as part of the second decomposition approach, they developed variations of a time window heuristic based on a decision theory approach for forming and sequencing the batches on a single machine.

Reichelt et al. (2006) were interested in minimizing total weighted tardiness and makespan at the same time($P_m|r_j$,batch, incomp.$|\Sigma w_j T_j$, C_{max}). In order to determine a Pareto efficient solution for the scheduling of jobs with incompatible families on parallel batch machines problem, they suggested a hybrid multi-objective genetic algorithm. They introduced a three-phase scheduling approach which contains a batch formation, a batch assignment, and a batch sequencing phase, respectively. They then characterized the NSGA-II metaheuristic that is used for the second phase. NSGA-II is generally used for multi-objective combinatorial optimization problems based on the principles of genetic algorithms. Furthermore, they also described the usage of a more advanced version of a hybrid multi-objective metaheuristic combining the NSGA-II algorithm and a local search technique in the assignment phase.

S/N	Title	Problem	Source	Year	Setup	Batch	Prec.	T W*	MO*	Solution Approach
1	Scheduling Jobs with Simple Precedence Constraints on Parallel Machines	Pm|rj,prec|$\Sigma w_j T_j^2$	Hoitomt et al	1990	0	0	1	0	0	Lagrangian relaxation technique and the list-scheduling
2	Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times	Pm|rj,batch, incomp |$\Sigma w_j T_j$	Monch et al	2005	0	1	0	0	0	Two and three Stage decomposition approaches, genetic algorithm, dispatching rules
3	Multiobjective Scheduling of Jobs with Incompatible Families on Parallel Batch Machines	Pm|rj,batch, incomp |$\Sigma w_j T_j$, Cmax	Reichelt et al	2006	0	1	0	0	1	Three-Phase scheduling approach hybrid multi-objective metaheuristic combining NSGA-II algorithm and a local search
4	Machine Learning Techniques for Scheduling Jobs with Incompatible Families and Unequal Ready Times on Parallel Batch Machines	Pm|rj,batch,incomp |$\Sigma w_j T_j$	Monch et al	2006	0	1	0	0	0	Apparent Tardiness Cost (ATC) dispatching rule
5	A Memetic Algorithm for Parallel Batch Machine Scheduling with Incompatible Job Families and Dynamic Job Arrivals	Pm|rj,batch,incomp |$\Sigma w_j T_j$	Cheng et al	2008	0	1	0	0	0	Memetic algorithm, BATC-II
6	Scheduling Jobs on Parallel Machines with Setup Times and Ready Times	Pm|rj,skj|$\Sigma w_j T_j$	Pfund et al	2008	1	0	0	0	0	Extension of the apparent tardiness cost with setups(ATCS) approach
7	A Fuzzy-Mixed-Integer Goal Programming Model for a Parallel-machine Scheduling Problem with Sequence-Dependent Setup Times and Release Dates	Pm|rj,skj| $\Sigma w_j T_j$, $\Sigma w_j C_j$	Gharehgozli et al	2009	1	0	0	0	1	Fuzzy mixed-integer goal programming(FMIGP)
8	Scheduling Jobs on Parallel Machines with Sequence-Dependent Setup Times, Precedence Constraints, and Ready Times Using Variable Neighborhood Search	Pm|rj,skj,prec|$\Sigma w_j T_j$	Driessel and Monch	2009	1	0	1	0	0	Variable neighborhood search

* TW Time Window, MO Multi objective 1 Considered, 0 Not considered

Table 3 Literature on the Pm|rj |ΣwjTj Problem

Gharehgozli et al. (2009) presented a new mixed-integer goal programming (MIGP) model for a parallel-machine scheduling problem with sequence-dependent setup times and release dates. Two objectives are considered in the model to minimize the total weighted flow time and the total weighted tardiness simultaneously($P_m|r_j,s_{kj}|\Sigma w_jT_j, \Sigma w_jC_j$). Due to the complexity of the model and uncertainty involved in real-world scheduling problems, they considered the problem under the hypothesis of fuzzy processing time's knowledge and two fuzzy objectives as the MIGP model.

Pfund et al. (2008) were interested in scheduling jobs with ready times on identical parallel machines with sequence dependent setups by minimizing the total weighted tardiness($P_m|r_j,s_{kj}|\Sigma w_jT_j$). Their approach is an extension of the Apparent Tardiness Cost with Setups (ATCS) approach by Lee and Pinedo (1997) to allow non-ready jobs to be scheduled. It allows a machine to idle for a high priority job arriving at a later time. To determine the scaling parameters for their composite dispatching rule (called ATCSR), they first developed a 'grid approach' that considers multiple values for the scaling parameters, generates multiple schedules, and chooses the best schedule for the solution. The experimentation was then used to develop regression equations to predict the values of the scaling parameters that would yield the highest quality solution.

Driessel and Mönch (2009) discussed a scheduling problem for jobs on identical parallel machines. Ready times of the jobs, precedence constraints, and sequence-dependent setup times are considered ($P_m|r_j,s_{kj},prec|\Sigma w_jT_j$). They are interested in minimizing the performance measure total weighted tardiness. They suggested a Variable Neighborhood Search (VNS) scheme for the problem. Based on computational experiments with stochastically generated test instances, they showed that the VNS approach clearly outperforms heuristics based on the Apparent Tardiness Cost with Setups and Ready Times (ATCSR) dispatching rule in many situations.

2.3 Rescheduling Literature Review

There is relatively little research on rescheduling, despite the extensive literature on scheduling problems. Vieira et al. (2003) and Aytug et al. (2005) provide extensive reviews of the rescheduling literature. Vieira et al. (2003) introduced definitions and a

classification for the rescheduling environments (the set of jobs that need to be scheduled), strategies (whether or not schedules are generated), policies (when rescheduling should occur), and methods (how schedules are generated and updated) presented in the rescheduling literature. Aytug et al. (2005) classified different approaches to scheduling in the presence of uncertainty into three groups: reactive scheduling, robust scheduling and predictive-reactive scheduling. There are three primary types of studies in rescheduling literature: studies on methods for repairing a schedule that has been disrupted, studies on methods for creating a schedule that is robust with respect to disruptions, and studies on how rescheduling policies affect the performance of the manufacturing systems (Viera et al., 2003).

Because of its practical importance, rescheduling has attracted a number of researchers in recent years. Bean and Birge (1991) considered a rescheduling problem with release dates and machine disruptions, and investigated "match-up" scheduling heuristics, which compute a transient schedule after a machine disruption. The objective is for the realized schedule to return to the predictive schedule within a certain time of the disruption occurring. Given this deterministic problem, they assumed that an optimal or near optimal pre-schedule had been constructed. Disruptions such as machine breakdown, resource unavailability, deviation in release dates, or unexpected new jobs introduced to the system. The match-up approach seeks to compensate for the disruption to minimize total tardiness. Their approach first fixes an initial match-up point and resequences the jobs on the disrupted machines to minimize the total tardiness cost. If the cost for meeting the selected match-up point exceeds a predetermined threshold, the match-up point is then incremented by some value, until the match-up point reaches a predetermined maximum value. A match-up is therefore possible only if there is enough idle time existing in the original schedule.

Church and Uzsoy (1992) considered the problem of minimizing maximum lateness on single-stage production systems involving single and identical parallel machines, where the only source of uncertainty is random job arrivals. They provided a rough taxonomy of existing approaches beginning with two extremes. Continuous rescheduling approaches take rescheduling action each time an event that is recognized by the system occurs. Periodic rescheduling, on the other hand, defines a basic time interval between rescheduling actions during which rescheduling actions are not permitted. Rescheduling actions are taken at periodic time points. These points

in time where rescheduling may be performed are referred to as rescheduling points. Any events occurring between rescheduling points are ignored until the following rescheduling point. Finally, they define event-driven rescheduling, in which a rescheduling action can be initiated upon the recognition of an event with potential to cause significant disruption to the system. Both continuous and periodic rescheduling can be viewed as special cases of event-driven rescheduling. They developed worst-case error bounds for the periodic approach assuming that an optimal algorithm is used to schedule the jobs available at each scheduling point. They then explored the performance of a combined periodic and event-driven approach, where additional rescheduling beyond what takes place at the rescheduling points can be caused by the arrival of a job with a tight due date.

Wu et al. (1993) proposed a composite objective to revise the schedules by continuous rescheduling approach for a single machine problem. They used a bicriterion approach, where the two conflicting objectives were minimizing the makespan and minimizing the deviation from the original schedule. Two sets of local search heuristics were developed; the first set used pairwise swapping methods with weighted combination of the two objectives. The second is a local search heuristic based on a genetic algorithm approach, considering two dimensional space of makespan and deviation; the quality of any given solution point in the space can be measured based on its Euclidian distance to the origin.

Unal et al. (1997) considered a single machine with newly arrived jobs that have setup times that depend on their part types. They proposed inserting new jobs into the original schedule to minimize the total weighted completion time or makespan of the new jobs without causing existing jobs to miss their deadlines. The due date performance of existing jobs is considered a priority, and is imposed as a constraint on the secondary criterion of the total completion time of the new jobs. Their approach does not incorporate a measure of schedule disruption.

Jain and Elmaraghy (1997) developed reactive scheduling mechanisms to reschedule a flexible manufacturing system. The triggers for rescheduling are the arrival of new jobs, order cancellations, changes in order priority, and machine failures.

Akturk and Gorgulu (1999) developed rescheduling procedures for machine failures that are designed to match-up with a long-term original schedule. They considered to

minimize the tardiness of the jobs as well as the match-up point in order to ensure schedule's stability. The authors defined the match-up point as the point in time at which the revised schedule merges again with the original schedule such that the pre-schedule can be followed again. After a machine breakdown, a match-up point for each machine is determined and part of the initial schedule that covers the time interval between the disruption and the Match-up point is rescheduled.

Vieira et al. (2000) presented analytical models for unrelated parallel machine scheduling to predict the performance measures for rescheduling strategies and quantify the trade-offs between performance measures. They considered dynamically arriving jobs and setups occur when production changes from one job type to another. In addition to periodic and event-driven rescheduling strategies, hybrid strategy based on queue size were studied. Under a hybrid strategy, rescheduling occurs not only periodically, but also whenever a pre-specified event occurs. The main performance measures of interest are the average flow time, machine utilization and setup frequency.

Alagoz and Azizoglu (2003) developed procedures for rescheduling identical parallel machines with minimizing the flow time as an objective where the random event that promoted rescheduling is a change in the machine eligibility constraints. Azizoglu and Alagoz (2005) considered a rescheduling problem in an identical parallel-machine environment with the disruption that one of the parallel machines down and the original schedule has to be updated to recover the effects of the disruption. In both works, they presented an optimizing algorithm for minimizing the stability measure subject to the constraint that the efficiency measure is at its minimum level. As an efficiency measure, they considered the total flow time and as a stability measure they considered number of disrupted jobs where a disrupted job is the one that is processed on different machines in the initial and revised schedules. They presented a branch and bound algorithm for the hierarchical problem of minimizing number of disrupted jobs subject to the constraint that total flow time is kept at its minimum. They proposed three heuristic procedures: a polynomial time algorithm and two branch and bound based heuristic procedures, to generate a set of approximate efficient schedules with respect to the total flow time and number of disrupted jobs criteria.

Hall and Potts (2004) considered a single machine rescheduling problem with disruption in the original schedule caused by multiple newly arriving jobs. A set of original jobs has been scheduled on one machine to minimize a given cost objective, and then several sets of new jobs arrive unexpectedly. The trade-off between the scheduling cost and the disruption cost were considered where the processing times only become known upon the arrival. Disruption cost examined as a constraint and as a part of the objective function. Firstly, they minimized the schedule cost subject to a limit on the deviation from the original schedule. Secondly, they minimized a total cost objective, which includes both the original cost measure and the cost of deviation. They presented a polynomial algorithm if it exists for each problem.

Mason et al. (2004) examined three different strategies for rescheduling complex job shops, while investigating the efficacy of each strategy to minimize the total weighted tardiness.

Curry and Peters (2005), considered the identical parallel machine scheduling problem with stepwise increasing tardiness cost objectives, non-zero machine ready times, machine reassignment costs, and constraints that limit machine reassignments. They focused on the arrival of new jobs which is described as a simple and common form of schedule disruption. Schedule nervousness was defined as occurring when a scheduling procedure reassigns many planned operations to different machines or different start times. They examined the tradeoff between schedule nervousness and tardiness in both single period and multiple period dynamic problems.

Yang et al. (2006) developed a parallel insertion algorithm implemented with rescheduling criteria to minimize the makespan of identical parallel-machine problem with uncertain job arrival and sequence-dependent setup time. Furthermore, they provided the probabilistic model to estimate the makespan for the problem that the inter-arrival time of jobs is exponentially distributed. Makespan was estimated with the summation of the arriving time of the last arriving jobs, the average waiting time for jobs in queue, and the average service time for jobs. Since the probabilistic model was under FIFO rule which is a common and simple dispatching rule, the expected makespan generated by the probability model was regarded as a lower standard in performance comparison and was used to evaluate the superiority of the scheduling algorithm.

Lee et al. (2006) studied two machine scheduling problem with transportation considerations in a machine related disruption environment. Their main assumption is that jobs that are assigned to the disrupted machines and have not yet been processed can either be moved to other available machines for processing with additional transportation time and cost, or can be processed by the same machine after the disruption. Objective function contains the original cost function (total weighted completion time and weighted deviation cost) and possibly transportation costs and disruption cost (deviation from completion times). They provided either a polynomial algorithm to solve the problem optimally, or show its NP-hardness and presented a pseudo-polynomial algorithm to solve the problem optimally.

Yang (2007) considered rescheduling problems with newly arriving jobs. In order to reduce the negative impacts of disruptions to the original schedule, the processing times of the newly arriving jobs can be reduced at a cost, called time compression cost. They considered the added feature of available compression time, which allows reducing the new jobs' processing times to decrease the impact of disruptions on the schedule. The objective of the problem is to minimize total cost after rescheduling, which includes schedule disruption costs, time compression related costs, and a cost that depends on a traditional measure of schedule efficiency. For the weighted tardiness cost efficiency measure, they provided a heuristic based on very large scale neighborhood (VLSN) search.

Duenas and Petrovic (2008) presented a new predictive-reactive approach to identical parallel machine scheduling problem with material shortage and job arrival as an uncertain disruption. The approach developed is based on generating a predictive schedule using dispatching rules to minimize the makespan. The predictive schedule absorbs the effects of possible uncertain disruptions through adding idle times to the job processing times. The added idle time is equal to the approximated repair time needed to recover from a disruption during the processing of a certain job. A material shortage is described by the number of disruption occurrences and disruption repair period. These parameters are specified imprecisely and modeled using fuzzy sets. If the impact of a disruption is too high to be absorbed by the predictive schedule, a rescheduling action that results new instability objective besides the makespan, is carried out. Two rescheduling methods namely left-shifting and building new schedules have been applied. The instability is measured as the starting time

deviations between the predictive schedule and the reactive schedule. The results of the study showed that the developed model is flexible and can also be used when new jobs arrive.

Arnaout and Rabadi (2008) developed repair and rescheduling algorithms for the unrelated parallel machine environment. They introduced right shift repair, fit job repair, partial rescheduling, and complete rescheduling rules to handle different rates of breakdowns and delays. They evaluated these based on several performance measures to ensure the optimization of both the schedule efficiency and stability. The efficiency measure was schedule makespan and the stability measure was the number of shifted jobs from one machine to another and the time to match-up with the original schedule after the occurrence of a breakdown.

Cheng et al. (2009) examined identical multiprocessor scheduling problems with resource and timing constraints in dynamic real-time scheduling with new job arrivals. They introduced a GA-based approach with a feasible energy function to illustrate the timing and resource constraints. The deadline constraint for each job is imposed on the proposed system.

Itayef (2009) considered the bicriteria flow shop scheduling problem with new job arrivals and two objectives: the initial one and a disruption objective. The aim is to generate or approximate the set of efficient schedules. The procedure involves two stages. In a first step, given a fixed order of jobs a conventional heuristic is proposed to successively assign the jobs to the machines. In the second step, the simulated annealing is applied to optimise the order of the jobs. They implemented a multiobjective metaheuristic method, MOSA that applies a SA procedure at each phase to optimize a linear aggregation function of the two objectives.

In these researches, rescheduling is needed as a result of different disruptions and events such as new job arrivals some of which can be rush, order cancellations, machine failure, changes in order priority, change in ready times, processing time delays, rework due to quality problems, material shortage, operator absenteeism, tool unavailability, due date changes, job order amount changes. Although Subramaniam et al. (2005) mentioned 17 types of disruption, most of the rescheduling literature has focused on two main types of disruption: *machine breakdowns* and *the arrival of new jobs*. However, some studies have also considered other types of disruption such as

process time variation (Subramaniam and Raheja, 2003), order cancellations (Jain and ElMaraghy, 1997; Shi-jin et al., 2007), and due-date changes (or urgent jobs) (Jain and ElMaraghy, 1997; Subramaniam and Raheja, 2003). Li et al. (2000) identified and investigated four sources of production disruptions: incorrect work, machine breakdowns, rework due to a quality problem and rush orders.

Updating a schedule upon the occurrence of a disruption affects both the efficiency and the stability of the process. The effect on the efficiency is usually measured in terms of the percentage change in the scheduling objective criterion between the original and the revised schedules. Stability has often been measured in terms of *deviations in the starting times* of operations between the two schedules (Abumaizar and Svestka, 1997; Subramaniam and Raheja, 2003; Subramaniam et al., 2005). Abumaizar and Svestka (1997) developed more schedule stability measures and a rescheduling algorithm for job shops.

There are three rescheduling repair methods: regeneration, right shift rescheduling, and partial rescheduling (Vieira et al., 2003). The regeneration approach solves the problem from scratch for the remaining operations in the schedule, with the objective of optimising the original scheduling objective criterion, with no regards to the resulting disturbance (Raheja and Subramaniam 2002). Regeneration approach reschedules the entire set of operations not processed before the rescheduling point, including those not affected by the disruption. The main disadvantage is the excessive computational effort and unsatisfactory response time. Rescheduling at the arrival of a rush job is useful but more frequent rescheduling does not improve system performance significantly (Church and Uzsoy, 1992). On the other hand, right shift rescheduling postpones each remaining operation by the amount of time needed to make the schedule feasible without changing the defined sequences of jobs on the machines to minimize the deviation caused in the schedule.

Many schedule repair approaches that lie between these two extremes have been proposed in the literature. Partial rescheduling updates only the operations affected directly or indirectly by disruption. Match-up scheduling (Bean et al., 1991) is a type of partial rescheduling to return to the predictive schedule within a certain time of the disruption occurring. Abumaizar and Svestka (1997) proposed the affected operations algorithm (AOR) for the job shop rescheduling problem.

Researches on parallel machine rescheduling are summarized in Table 4. Church and Uzsoy (1992), Curry and Peters (2005), Yang et al.(2006), Duenas and Petrovic (2008), and Cheng et al. (2009) are the most related to AR rescheduling problem when machine type, disruption type, rescheduling environment, rescheduling strategy and rescheduling method issues are evaluated. These parallel machine rescheduling problems generally have multiple objectives: the objective of the original problem (e.g. minimization total weighted tardiness) and the minimization of the difference between the new schedule (after rescheduling) and the original schedule (before rescheduling). Disruption cost were examined as a constraint by maximizing the schedule performance subject to a limit on the disruption cost or as a part of a combined objective function. Polynomial time algorithms and approximate algorithms including dispatching rules and metaheuristic methods are developed by regarding both objectives to obtain qualified solutions in a reasonable time.

In conclusion, there is very little existing research on the ARSP for both inter-theater and in-theater operations, and there is a gap in the in the corresponding area of parallel machine scheduling and rescheduling with release time and due date-to-deadline windows to minimize the total weighted tardiness. In this research, time window between due date and deadline were firstly considered in parallel machine scheduling problem by defining a piecewise tardiness cost. The scheduling and rescheduling problem framed in this dissertation has never been tackled in the Operations Research literature. To fill this research gap, this dissertation will introduce new Mixed Integer Programming models to find optimal solutions as well as heuristic and metaheuristic algorithms to obtain near-optimal schedules efficiently for both the scheduling and rescheduling problems. All of the exact and approximate methods introduced will be applied for the first time to the problem at hand. Additionally, there is very little research addressing multi-objective optimization (MOO) in the parallel machine scheduling environments that are susceptible to job related disruptions. The components of the multi-objective formulated in this research are the total weighted tardiness which represents schedule's quality, and the proportion of rescheduled jobs that change machine assignment and/or starting time, which represents the schedule's stability. The component that measure schedule instability is typically the number (or proportion) of jobs that change machine assignment or the change in jobs' start times. In this dissertation, however, both

measures are combined into one measure that is introduced to the literature and combines the proportion of the rescheduled jobs due to change in machine assignment and/or change in starting time.

S/N	Title	Problem	Source	Year	M	Disruption	MO	Solution Approach
1	Match-up Scheduling with Multiple Resources, Release Dates and Disruptions	Pm|rj, incomp |$\Sigma w_j(T_j+F_j)$	Bean et al	1991		8 disruption types	0	Match-up approach, integer programming, Priority Rule Approach
2	Analysis of Periodic and Event Driven Rescheduling Policies in Dynamic Shops	1 and Pm|rj |L_{max}	Church and Uzsoy	1992	I	Job arrival	0	Hybrid event-driven approach with dispatching rules, worst case error bounds for simulation
3	Predicting the Performance of Rescheduling Strategies for Parallel Machine Systems	Pm|rj,skj| many performance measures(avg flow time , machine utilization, setup frequency)	Vieira et al	2000	U	Job arrival	1	Complete regeneration, simulation
4	Rescheduling Parallel Machines with Stepwise Increasing Tardiness and Machine Assignment Stability Objectives	Pm|rj, reassignment |$\Sigma\Sigma w_j$,U_j| (stepwise increasing tardiness cost, machine reassignment cost	Curry and Peters	2005	I	Job arrival	1	Branch and price algorithm, Simulation
5	A Comparative Study to Minimize the Makespan of Parallel-Machine Problem with Job Arrival in Uncertainty	Pm|rj,skj| Cmax	Yang et al	2006	I	Job arrival	0	Probabilistic parallel insertion algorithm, FIFO dispatching rule
6	An Approach to Predictive-reactive Scheduling of Parallel Machines Subject to Disruptions	Pm|rj, compatible| Cmax, completion time deviation	Duenas and Petrovic	2008	I	Material shortage, Job arrival	1	Fuzzy based predictive and reactive left-shifting and building new schedules methods
7	Rescheduling of Unrelated Parallel Machines under Machine Breakdowns	Pm|rj| ΔCmax, Number of shifted jobs, time to Match-up	Arnaut and Rabadi	2008	R	Machine breakdown	1	Right shift repair, fit job repair, partial rescheduling, and complete rescheduling, MIP
8	Dynamic Hard-real-time Scheduling using Genetic Algorithm for Multiprocessor Task with Resource and Timing Constraints	Pm| multiprocessing, preemptive, no migration, energy function |	Cheng et al	2009	I	Job arrival	0	Reactive scheduling, GA-based approach, dynamic real-time scheduling

M machine type (I Identical, U Uniform, R Unrelated) MO Multi Objective 1 Considered, 0 Not Considered

Table 4 Parallel Machine Rescheduling Literature

CHAPTER 3

AERIAL REFUELING SCHEDULING PROBLEM METHODOLOGY

The ARSP is denoted by $P_m|r_j,\ d\text{-}to\text{-}D\ window|\Sigma w_jT_j$. It represents a system with m identical machines in parallel, job j arrives at ready time r_j and should leave by the due date d_j and strictly before the deadline D_j. ARSP has also the following assumptions: Any job can be processed on at most one machine at any time. No preemption is allowed; i.e., once an operation is started, it is continued until complete. Machines are always available (no breakdowns). Each machine can process at most one job at any time. Setup times are sequence-independent and are included in processing times. Processing times and technological constraints are deterministic and known. Since higher wing availability is desired and the relative importance levels of the wings affect the scheduling in ARSP, the total weighted tardiness is used as a scheduling performance measure in the mathematical model and approximate algorithms.

In this chapter, a mixed integer linear programming model (MILP) is firstly developed to find optimal solutions for ARSP. However, since the identical parallel machine scheduling is NP-hard even with only two machines (Karp 1972; Garey and Johnson 1979; Blazewicz et al., 2007), ARSP is also NP-hard. Consequently, it is required to develop qualified, easily, and quickly implemented solution approaches with reasonable computation times.

3.1 Mathematical Model

There is more than one type of integer programming formulation to model the problem addressed here based on variables such as time–indexed, linear ordering, and positional assignment for parallel machine scheduling.

Time–Indexed Formulation: The model is formulated with variables indexed by pairs *(j, t)* where *j* denotes a job and *t* a time period. In order to obtain a finite number of variables, a fixed time horizon *T* is introduced and time is divided into periods *1, 2...., T* where period *t* starts at time *t-1* and ends at time *t*. A decision variable is equal to one, if

job j is started in period t; otherwise it is equal to zero. Related work includes Liaw et al. (2003), Tanaka and Araki (2008) and Paula et al. (2010).

Linear Ordering Formulation: The set of feasible solutions is considered as (a subset of) the set of permutations or linear orderings of the job set. Variables that characterize linear orderings are used to describe the feasible solutions. A decision variable is equal to one if job j precedes i in machine k, otherwise it is equal to zero. Related work includes Logendran and Subur (2004), Tang et al. (2007), Biskup et al. (2008) and Gharehgozli et al. (2009).

Positional Assignment Formulation: This formulation works with decision variables that refer not to the original jobs, but to their positions in the schedule. A decision variable is equal to one if job j is assigned to position k of machine i, otherwise it is equal to zero. An example of a related work is by Tavakkoli-Moghaddam et al. (2009).

A mixed integer linear programming model (MILP) is developed to find optimal solutions for ARSP using both *linear ordering and assignment decision variables*. The goal in the ARSP is to find optimal schedules of wings to identical parallel tankers to minimize the total weighted tardiness.

3.1.1 Notations

Jobs have a piecewise tardiness cost function as follows (see Figure 6):

$$\text{Tardiness, } T_j = \begin{cases} 0, & \text{if } C_j \le d_j; \\ C_j - d_j, & \text{if } d_j < C_j \le D_j; \\ F, & \text{if } D_j \le C_j \end{cases}$$

In order to represent the starting point for each machine, a dummy index 0 is defined for predecessor of the first job.

Indices

$i = 0,1,2,3, \ldots,$ *n predecessor jobs;*

$j = 1,2,3, \ldots,$ *n successor jobs;*

$k = 1,2,3, \ldots,$ *m machines;*

Parameters

p_j = processing time of job j;

r_j = release (ready) time of job j;

w_j = weight of job j;

d_j = due date of job j;

D_j = deadline of job j;

M = a large positive integer.

F= fixed cost of returning back to base. It must be much larger than $D_j - d_j$ values.

Decision Variables

C_j = completion time of job j;

T_j = piecewise tardiness of job j;

$$x_{ijk} = \begin{cases} 1, & \text{if job } i \text{ precedes job } j \text{ on machine } k; \\ 0, & \text{otherwise.} \end{cases}$$

$$y_{jk} = \begin{cases} 1, & \text{if job } j \text{ is assigned to machine } k; \\ 0, & \text{otherwise.} \end{cases}$$

3.1.2 A Mixed Integer Programming Formulation

Objective : Minimize Total Weighted Tardiness

$$\min Z = \sum_{j=1}^{n} w_j T_j$$

Constraints

1. Some jobs cannot be assigned to a machine (returning back to base) and no job can be assigned to more than one machine.

$$\sum_{k=1}^{m} y_{jk} \leq 1 \quad j = 1,...,n$$

2. You cannot have more than one job in the first position at a certain machine and it is possible for a machine not to any jobs assigned to it at all.

$$\sum_{j=1}^{n} x_{0jk} \leq 1 \quad k = 1,...,m$$

3. If job i precedes job j on machine k, then job j must be scheduled after it becomes ready and after the completion of its predecessor job i (hence the term $max(r_j, C_j)$).

$$C_j + M(1 - x_{ijk}) \geq \max(r_j, C_i) + p_j \quad i = 0,1,....,n \quad j = 1,...,n \quad k = 1,...,m$$

This nonlinear constraint can be formulated alternatively by two linear constraints:

3.a. If job j is assigned to a machine, then job j must be scheduled after its release time.

$$C_j + M(1 - \sum_{k=1}^{m} y_{jk}) \geq r_j + p_j \quad j = 1,...,n$$

3.b. If job i precedes job j on machine k, then job j must be scheduled after the completion time of its predecessor job i.

$$C_j + M(1 - x_{ijk}) \geq C_i + p_j \quad i = 1,....,n \quad j = 1,...,n \quad k = 1,...,m$$

4. If job j is assigned to one of the machines, then its completion time cannot exceed the deadline. If job j is not assigned to machine k, then completion time of job j will be zero.

$$C_j \le D_j . \sum_{k=1}^{m} y_{jk} \qquad j = 1,...,n$$

5. If job j is assigned to machine k, then job j must be preceded by a job i (includes starting dummy job 0). Job j can be first assigned job to a machine or be preceded.

$$\sum_{i=0, i \neq j}^{n} x_{ijk} = y_{jk} \qquad j = 1,...,n \quad k = 1,...,m$$

6. If job i is assigned to machine k, then job i can be succeeded by at most one job j or be the last job.

$$\sum_{j=1}^{n} x_{ijk} \le y_{ik} \qquad i = 1,...,n \quad k = 1,...,m$$

7. If job i precedes job j on machine k, then job j cannot proceed job i.

$$x_{ijk} + x_{jik} \le 1 \qquad i = 1,...,n \quad j = 1,...,n \quad k = 1,...,m$$

8. Piecewise tardiness definition can be formulated linearly by constraints (8.a, 8.b, 8.c):

   8a. Tardiness is equal to or larger than the lateness.

   $$T_j \ge C_j - d_j \qquad j = 1,...,n$$

   8b. Tardiness is equal to the return back penalty if job j is not assigned to any machine.

   $$T_j \ge F(1 - \sum_{k=1}^{m} y_{jk}) \qquad j = 1,...,n$$

   8c. Nonnegative completion time and tardiness values.

   $$C_j, T_j \ge 0 \qquad j = 1,...,n$$

9. Binary variables.

$$x_{ijk}, y_{jk} = binary \quad i = 0,1,...,n \quad j = 1,...,n \quad k = 1,...,m$$

The complete Mixed Integer Linear Programming (MILP) model can then be laid out as follows:

$$\min Z = \sum_{j=1}^{N} w_j T_j$$

s.t.

$$\sum_{k=1}^{m} y_{jk} \leq 1 \quad j = 1,...,n \tag{1}$$

$$\sum_{j=1}^{n} x_{0jk} \leq 1 \quad k = 1,...,m \tag{2}$$

$$C_j + M(1 - \sum_{k=1}^{m} y_{jk}) \geq r_j + p_j \quad j = 1,...,n \tag{3a}$$

$$C_j + M(1 - x_{ijk}) \geq C_i + p_j \quad i = 1,....,n \quad j = 1,...,n \quad k = 1,...,m \tag{3b}$$

$$C_j \leq D_j . \sum_{k=1}^{m} y_{jk} \quad j = 1,...,n \tag{4}$$

$$\sum_{i=0, i \neq j}^{n} x_{ijk} = y_{jk} \quad j = 1,...,n \quad k = 1,...,m \tag{5}$$

$$\sum_{j=1}^{n} x_{ijk} \leq y_{ik} \quad i = 1,...,n \quad k = 1,...,m \tag{6}$$

$$x_{ijk} + x_{jik} \leq 1 \quad i = 1,...,n \quad j = 1,...,n \quad k = 1,...,m \tag{7}$$

$$T_j \geq C_j - d_j \quad j = 1,...,n \tag{8a}$$

$$T_j \geq F(1-\sum_{k=1}^{m} y_{jk}) \quad j=1,...,n \tag{8b}$$

$$C_j, T_j \geq 0 \quad j=1,...,n \tag{8c}$$

$$x_{ijk}, y_{jk} = binary \quad i=0,1,...,n \quad j=1,...,n \quad k=1,...,m \tag{9}$$

Optimization Programming Language (OPL) Studio 6.3 was used to implement this model and CPLEX 12.1 was used to solve it. Input to the OPL model are indices (*predecessor jobs, successor jobs, and machines*), parameters (*release times, processing times, weights, due dates, and deadlines*) and fixed unavailability cost *F*. Optimal solutions are discussed in Chapter 4.

3.2 Review for Solution Methods

In scheduling problems, generally, exact algorithms and approximate algorithms can be used to find the optimal or the best solutions. First, exact algorithms are guaranteed to find an optimal solution for every finite size instance of a combinatorial optimization problem in bounded time. Second, in order to yield a fast and reasonable solution to a problem, approximate algorithms which are methods based on the experience or judgement, can be used. They can be more quickly developed and used than optimization routines. But they cannot be guaranteed to produce the mathematically optimal solution.

3.2.1 Exact Algorithms

The most common exact/complete methods for scheduling problems are branch and bound algorithms, mixed integer programming, and decomposition methods. However, since ARSP is NP-hard, obtaining optimal solutions for large instances will be computationally difficult. Therefore, it becomes necessary to develop qualified approximate solutions in reasonable computational times.

3.2.2 Approximate Algorithms

Generally, approximate algorithms can be classified as constructive, local search, and metaheuristic algorithms.

3.2.2.1 Constructive Algorithms

Constructive algorithms generate solutions by gradually adding parts of the solution to the initially empty partial solution. Their major advantage is in computational time requirements as largely being the fastest approximate algorithms. However, they have generally inferior quality solutions when compared to local search techniques and some special implementations need high computational load. The most widely used constructive heuristics for scheduling problems are the various dispatching rules. Dispatching (or Priority) Rules are the most common heuristics for scheduling problems due to their easy implementation and low requirements in computational power. They are designed so that the priority index for each job can be computed easily using the information available at any time. They are useful to find a reasonably good schedule with regard to a single objective (Pinedo, 2008). Moreover, some priority rules generate the optimum solution in certain simple problems (e.g. the minimization of flowtime in single-machine scheduling where the SPT Priority rule generates the global optimum solution). Although they perform very well in certain cases, no rule exists that can be applied to all problems and perform satisfactorily and there is no way to estimate the performance of a dispatching rule for a specific instance *a priori* (Zobolas et al., 2008).

A number of simple dispatching rules such as the Earliest Due Date (EDD), Shortest Processing Time (SPT), Minimum SLACK (MSLACK), and Slack per Remaining Processing Time (S/RPT) have been applied to solve total tardiness, weighted tardiness, and maximum tardiness related problems (Lee and Pinedo, 1997).

A number of composite dispatching rules, made up of attributes and some scaling parameters, have also been developed for different types of machine scheduling environments (Logendran and Subur, 2004). Vepsalainen and Morton (1987) applied the principles of the Cost Over Time (COVERT) and the Modified Operation Due date

(MOD) rule to develop the apparent tardiness cost heuristic (ATC) for the parallel machine total weighted tardiness problem. Rachamadugu and Morton (1982) proposed the RM heuristic for parallel machine scheduling. Morton and Pentico (1993) modified the RM heuristic to allow consideration of jobs arriving at a later time and develop their X-RM heuristic (Pfund et al., 2008).

3.2.2.2 Local Search Algorithms

Local search algorithms start from an initial solution (most of the times generated by a constructive heuristic or randomly) and iteratively try to replace part or the whole solution with a better one in an appropriately defined set of neighboring solutions. For every defined neighborhood of solutions, the solution or solutions of highest quality (i.e. the best objective function value) are called locally optimum solutions within the defined neighborhood. In order to replace parts of an initial solution, local search methods perform a series of moves leading to the formation of new solutions in the same neighborhood. The main drawback of basic local search methods is that they get easily trapped in local optima as they are myopic in nature. More specifically, local search with appropriate moves can be very effective in exploring a neighborhood of an initial solution but no mechanism exists that can lead to other distant neighborhoods of the solution space where the global optimum may exist. To remedy this weakness, new modern local search methods (explorative local search) have been developed with embedded meta-strategies to guide the search process (Zobolas et al., 2008).

3.2.2.3 Metaheuristic Algorithms

A metaheuristic is a higher level heuristic procedure designed to guide other methods or processes towards achieving reasonable solutions to difficult combinatorial mathematical optimization problems (Silver, 2002). The aim of the new methodology is to efficiently and effectively explore the search space driven by logical moves and knowledge of the effect of a move facilitating the escape from locally optimum solutions. This is achieved by either allowing worsening moves or generating new starting solutions for the local search in a more intelligent way than just providing random initial solutions They guide

the search through the solution space, basically combining some form of heuristic methods and usually local search in higher level frameworks.

Starting from an initial solution built by some heuristic, metaheuristics improve the solution iteratively until a termination criterion is met. They exactly or approximately evaluate candidate solutions, maintain a record of the best solution obtained so far, and have a mechanism for termination (a certain total number of iterations or a prescribed number of consecutive iterations without any improvement). The termination criterion can be elapsed time, number of iterations, number of evaluations of the objective function, and so on (Blum and Roli, 2003).

Metaheuristic methods have an advantage over simpler heuristics in terms of solution robustness; however they are usually more difficult to implement and tune as they need special information about the problem to be solved to obtain good results (Zobolas et al., 2008). They need settings of parameters that affect the search process significantly (the so-called cooling parameter in simulated annealing and the length of the tabu list in tabu search) (Lee et al., 1997).

Generally, the most widely applied metaheuristics are simulated annealing, tabu search, genetic algorithms, and ant colony optimization are reviewed in the following.

Simulated Annealing (SA)

Simulated Annealing (SA) is one of the most frequently used metaheuristics to find near optimal solutions to combinatorial optimization problems through the process of probabilistic state transition (Silver, 2002). It is based on the work of Metropolis et al. (1956) who simulated the energy levels in cooling solids by producing a sequence of states. SA is a local search improvement heuristic that generates neighbor solutions by simple moves from the current solution starting with a randomly generated initial solution. The SA algorithm generates a new solution in the neighborhood of a current (or initial) solution. The objective function value of the new candidate solution is then compared to the objective function value of the current solution. If the new objective function value is better than the value of the current solution, the new solution is

automatically accepted. If the objective function value is worse than the value of the current solution, then the candidate solution may still be accepted with some probability known as the probability of acceptance Pa = exp (-Δ/T) where Δ is the difference between objective values of the candidate and current solution and T isa temperature control parameter of the SA algorithm. This temperature is periodically reduced after allowing the SA to run for a certain number of iterations and explore candidate solutions in the neighborhood of the current solution. When the temprature is low enough, the SA converges (freezes) to a final solution. What makes SA more effective than greedy and local search methods is that it has a mechanism that prevents getting trapped at local optima by allowing the acceptance of worse solutions with a certain probability (Hazır et al., 2008). The SA method was chosen in this research as a robust meta-heuristic to provide immediate results to the problem. SA can deal with arbitrary systems and cost functions and is relatively easy to implement even for complex problems. It generally gives a good solution and statistically guarantees finding an optimal solution, but cannot tell whether it has found an optimal solution (Tan et al., 2001).

Tabu Search (TS)

Tabu search (TS) is a local-search, improvement heuristic similar to SA with a punishment mechanism to escape from a local optimum by forbidding or penalizing moves that cause cycling among solution points previously visited (Hazır et al., 2008). The main strength of tabu search is the use of memory that speeds up the solution space search process (Zobolas et al., 2008). The search process keeps track of a so-called tabu tenure with a fixed number of entries, called the size (or length) of the list. This size is a key controllable parameter of the metaheuristic.

Every time the search performs a mutation in order to go from one schedule to a neighboring schedule, the reverse mutation is placed at the top of the tabu list. All other entries on the list are pushed down one position and the entry at the bottom is deleted. The reason for keeping such a tabu list is based on the fact that it is not desirable to allow a search in a subsequent move to return to a schedule already considered. Since the most basic form of tabu search is a deterministic process, there is always the danger of *cycling*.

The cycling phenomenon depends very much on the length of the tabu list. If the length of the tabu list is too small, the process may have a high likelihood of cycling. If the length of the list is chosen too large, then the freedom of the search process is curtailed and the process may be less likely to find good solutions (Lee et al., 1997).

Genetic Algorithm (GA)

A Genetic Algorithm (GA) is a population-based improvement heuristic based on natural selection and evolutional theory. The GA approach, first developed by John Holland (1975), is based on the principles of evolution and genetics to guide the search according to ability of the individuals to survive in the competitive environment. A solution is represented by knowledge structures (also called chromosomes) that are composed of genes in the form of a binary or nonbinary string. The value of the feature associated with a particular gene is called its allele. A collection of chromosomes is called a population. Genetic algorithms work with a population of solutions that at each iteration each solution in the current population is evaluated.

The evaluations serve to select a subset of the solutions to be either used directly in the next population or indirectly through some form of transformation or variation. The next population is formed in two steps. First, recombination or a cross-over operator considers two individuals in the population and create a new individual by combining one part of one chromosome with another part of the other chromosome (Lee et al., 1997). The role of the crossover operator is to form new fit individuals from fit parents to inherit the genetic material of the parent chromosomes (natural selection). Second, a mutation operator alters one or more components of a selected individual to provide new information into the knowledge base. The mutation operator (typically with a very small probability) serves as a secondary search that ensures that all points in the search domain are reachable by introducing new genetic material into the population (Silver, 2002). The incumbent solution is expected to improve as populations are generated until a stopping criterion is reached (Hazır et al., 2008).

Ant Colony Optimization (ACO)

The ant colony optimization (ACO) algorithm is a constructive and population-based metaheuristic inspired by the collective behavior of real ants for the survival of their colonies (Hazır et al., 2008). Ants deposit pheromone on their path to the food sources and the ability of other ants to smell this chemical enables them to find the shortest path between their nest and the food. When more ants collectively follow a trail, the trail becomes more attractive for being followed in the future. The capability of a single ant to locate food is limited, but the collected/shared knowledge helps the colony to find efficient paths to a food source; the information about the good paths is passed on to the members of the colony via the amount of pheromone deposited on the paths.

Dorigo and Gambardella (1997) used this feature in solving combinatorial optimization problems. Each ant constructs a solution by moving from one state to another adjacent state by applying a stochastic local search policy (Hazır et al., 2008). Each (artificial) ant probabilistically selects the solution components of the problem based on the pheromone trail and heuristic information in each iteration. The heuristic information is the desirability between the solution components which is defined by the structural properties of the concerned problem (Raghavan and Venkataramana, 2009). A tour ends when all the ants of the colony generate solutions of different quality. The information gathered at the end of each tour is updated through a global pheromone updating rule. The ants are expected to generate better solutions by using this information in the next tour. The algorithm terminates when a stopping criterion is satisfied (Hazır et al., 2008). The use of a colony of ants can give the algorithm increased robustness and in many Ant Colony Optimization (ACO) applications the collective interaction of a population of agents is needed to efficiently solve a problem. ACO can be applied to any discrete optimization problem for which some solution construction mechanism can be conceived (Dorigo and Stützle, 2005).

Comparison of the Metaheuristics

The choice of which metaheuristic approach to use depends on some factors including the decision area, the frequency of the decision made, available time to develop, the

analytical qualifications of the decision maker, the problem size, presence of significant stochastic elements. However, it is desirable to have very good *average* performance on average and robustness in terms of low chance of achieving a poor solution and insensitive performance to the actual or estimated values of the parameters of the problem. For sensitive results, conditions under which the heuristic should be used has to be specified (Silver, 2002).

A review for comparative studies on the applications in the scheduling problems may be useful to have a rational approach to perform the metaheuristics for effective ARSP solutions. Metaheuristic algorithms can be divided in population based and single point search. Population-based metaheuristic methods (GA, ACO) combine a number of solutions in an effort to generate new solutions that inherit merits of the old ones. On the other hand, single point search methods (SA, TS) improve upon a specific solution by exploring its neighborhood with a set of moves. Another important and widely used classification scheme is based on the memory used during the search process. Memory usage (in TS, GA and ACO) constitutes a main characteristic of effective metaheuristic methods and is the indication of the intelligence employed during the search process. Memory-less algorithms (SA) are rarely employed for complex combinatorial optimization problems (Zobolas et al., 2008).

For the both single point search methods, SA and TS, the neighborhood design as well as the search pattern within a neighborhood can be the same. Both start with an initial complete feasible solution and iteratively generate additional solutions (Silver, 2002). The time needed to implement SA or a simple TS is attractively short. Both SA and TS implementations for a given problem can usually be adapted to take into account constraints which were not in the initial formulation of the problem (Pirlot, 1996). In most comparisons of simple TS with SA, TS has generally been found superior, mainly by being able to provide solutions of comparable quality in much shorter time. It is often possible to obtain solutions of similar quality with both but TS generally runs much faster. On the contrary, even a simple TS involves more tactical choices and hence needs slightly more time to be implemented and tuned (Pirlot, 1996). Another difference between simulated annealing and tabu search is about the acceptance rejection technique.

First, a tabu search permits moving away from a local optimum (i.e. diversifying) by an essentially deterministic mechanism, whereas, a probabilistic device is used in simulated annealing. Second, a tabu search tends to temporarily permit moving to poorer solutions only when in the vicinity of a local optimum, whereas this can happen at any time in simulated annealing (Silver, 2002). Third, the basic form of a tabu search does not include any random elements in contrast with simulated annealing (Silver, 2002).

Tabu search and SA may be considered as special forms of genetic algorithms with the number of individuals in each generation equal to one. The fact that genetic algorithms keep track of multiple solutions at each iteration may make them more powerful (but at the same time slower) than simulated annealing and tabu search. GAs can deal with wide variety of problems and can work with other heuristics (hybrid GA). GAs work with a coding of the parameter set, and not necessarily the parameters themselves. They do not guarantee optimal solutions, but they can generally find good solutions relatively quickly. The design of the neighborhood in GA is different from those in simulated annealing and tabu search In the diversification scheme of genetic algorithms, the result of each iterative step is a number of different solutions and all are carried over to the next step (in simulated annealing and tabu search, only a single solution is transferred from one iteration to the next). In genetic algorithms, the neighborhood concept is therefore not based on a single solution, but rather on a set of solutions. A new solution can be constructed by combining different parts from different schedules within the set (Lee et al., 1997). Genetic algorithms fail to intensify the search to the most promising regions of a neighborhood. Thus, the successful implementations of genetic algorithms usually incorporate a local search procedure for search intensification (Zobolas et al., 2008).

Besides these theoric comparison, there are also some experimental studies comparing the most common metaheuristics. Della Croce et al. (1992) made a comparison of various techniques, including the shifting bottleneck technique, tabu search and genetic algorithms. In their comparison, genetic algorithms appeared to be the least effective technique among the three neighborhood search techniques.

Kim et al. (1996) proposed several algorithms, including tabu search and simulated annealing methods with the objective of minimizing mean tardiness. After an

experimental study to set the best values for the parameters, four tabu search methods and four simulated annealing methods, all starting from a given initial solution, were presented. The simulated annealing algorithms (considering insertion neighborhood) gave the best results outperforming the remaining heuristics and metaheuristics. However, the results obtained with the simulated annealing algorithms (considering the interchange neighborhood) were the worst among all metaheuristics.

Jozefowska et al. (1998) presented three metaheuristics (SA) (TS) and (GA) to find solutions for some discrete/continuous scheduling problems. All the described algorithms were designed, adjusted, and applied to the considered class of scheduling problems. Comparing the performance of metaheuristics tested in the experiment, the TS performed best, finding the largest number of optimal solutions and showing smallest deviation from optimum for all the problem sizes.

Two simulated annealing algorithms which had a difference in the perturbation scheme were proposed in Parthasarathy and Rajendran (1998). The initial solution for both methods was given by a specific rule. Results were compared against the other heuristics including tabu search and two simulated annealing methods produced the best results.

Hasija and Rajendran (2004) presented a simulated annealing method to minimize total tardiness. The initial solution was given by a specific rule proposed in Parthasarathy and Rajendran (1998) which was improved through a perturbation scheme. Then, the simulated annealing was applied to improve this initial solution. The performance of the proposed algorithm was evaluated against the tabu search and the simulated annealing. The results showed that the simulated annealing algorithm obtained better results than the other two methods.

Vallada et al. (2008) gave an extensive and comprehensive review of heuristic and metaheuristic methods for the permutation flowshop scheduling problem with the objective of minimising the total tardiness. Two simulated annealing algorithms outperformed all the other methods evaluated. They showed that the tabu search methods are good metaheuristics for the interested problem.

Hazır et al. (2008) compared the performances of SA, TS, GA, and ACO metaheuristics on the Customer Order Scheduling Problem. According to their results, the output quality of a metaheuristics were depending on the problem size, as the complexity of the problem reduced, all four methods performed better, but among them, the best algorithm was SA, even though it is the least intelligent one. However, for dense problem instances, TS and ACO outperformed SA and GA. TS had slightly better results than ACO. TS and ACO converged to their best results faster than SA and GA. Therefore in case of time limitations, the authors concluded that TS and ACO were more preferable and with respect to the number of parameters that have to be fine tuned, implementation of TS or SA was easier than implementation of the others.

Jungwattanakit et al. (2009) investigated both constructive and iterative (SA, TS and GA-based algorithms) approaches for minimizing a convex combination of makespan and the number of tardy jobs for the flexible flow shop problem with unrelated parallel machines and setup times. For the recommended SA, TS, and GA parameters, they investigated the performance of the algorithms with random initial solutions. They found that the SA-based algorithm outperformed the other algorithms. Then, they studied the influence of the initial solution on these algorithms. The results showed that the SA-based algorithms were still recommendable (Jungwattanakit et al., 2009). However, simulated annealing was unable to quickly achieve good solutions to job shop scheduling problems, perhaps because it is a generic and memory-less technique (Jain and Meeran, 1998).

As a result, the initial solution, the setting of parameters, the language, and manner in which the procedure is coded, the platform on which the study is conducted affect the outcome of each comparative study. But it is clear that simulated annealing is easy to implement while obtaining qualified solutions.

3.2.2.4 Metaheuristic for Randomized Priority Search (MetaRaPS)

MetaRaPS was initially introduced by DePuy et al. (2001) based on the idea of COMSOAL (Computer Method of Sequencing Operations for Assembly Lines), which is a computer heuristic designed by Arcus (1966) to solve the assembly line balancing problem.

Moraga et al. (2006) defines MetaRaPS as "generic, high level search procedures that introduce randomness to a construction heuristic as a device to avoid getting trapped at a local optimal solution". MetaRaPS combines the mechanisms of priority rules, randomness, and sampling. The main properties of MetaRaPS are the use of randomness over a construction heuristic as a mechanism to avoid local optima, and the expectation that good unimproved solutions lead to better neighboring solutions (Lan et al. 2007). DePuy et al. (2001) expresses that run times for MetaRaPS is not significantly affected by the size of the problem, it is easy to understand and to implement, and can generate a feasible solution at every iteration.

MetaRaPS is a general form of other greedy algorithms, COMSOAL and GRASP (Greedy randomized Adaptive Search Procedure; Feo and Resende, 1995) and it is more flexible and more efficient (Lan et al. 2007). MetaRaPS is a two-phase iterative search procedure with a constructive phase to create feasible solutions through randomized construction heuristic priority rules, followed by an improvement phase to improve them at each iteration.

In the constructive phase, a solution is built by repeatedly adding basic feasible elements or activities to the current solution based on priority rules until a stopping criterion is satisfied. Each feasible basic element is characterized by a greedy score according to some priority rule. The best feasible element might assume either the lowest score or the highest score depending on the definition of the priority rule (Lan et al. 2007). Generally, solutions obtained by implementing only constructive algorithms can reach mostly local optima. To avoid local optima, MetaRaPS does not select the component or activity with the best priority value every time; instead, the algorithm may accept one with a good priority value, not necessarily the best, based on a randomized approach (DePuy et al. 2005). This randomness of MetaRaPS introduces diversification to the process while keeping the same quality produced by the priority rule (Rabadi et al., 2006).

A MetaRaPS algorithm uses four parameters: the number of iterations (I), the priority percentage (p%), the restriction percentage (r%), and the improvement percentage (i%). First, the number of iterations (I) determines the number of feasible solutions constructed. Second, the parameter p% is employed to decide the percentage of time, the component,

or activity with the best priority value will be added to the current partial solution, and 100%-*p*% of time the component or activity with the good priority value is randomly selected from a candidate list (CL) containing "good" components or activities. Third, the CL of components or activities with good priority values is created by including ones whose priority values are within *r*% of the best priority value. The smaller %p and the larger %r for a given priority rule, the more randomness will be introduced. Elements are added to the solution until a feasible solution is generated. The construction stage ends after a feasible solution is generated.

The improvement phase is performed if the feasible solutions generated in the construction phase are within *i*% of the range between the best and worst unimproved solution value obtained in the construction phase (Moraga et al., 2006). In MetaRaPS, only the constructed solutions with promising, or good enough, values are improved by using a local search algorithm.

3.2.2.5 Integrating Metaheuristics and Dispatching Rules

An effective heuristic may be combined with a metaheuristic in order to quickly arrive to a solution close to the optimal. There is an advantage in starting from a better initial solution than a random solution in problems where good solutions cannot be easily obtained through a small number of elementary transformations of a random solution (Pirlot, 1996). Moreover, a good seed solution can decrease the computational time considerably.

There are some researches as an example of this form of improving an initial solution. In Lee and Pinedo (1997), a three phase heuristic was presented for scheduling jobs on parallel machines with sequence-dependent setup times to minimize the total weighted tardiness. In the first phase, factors or statistics which characterize an instance were computed. The second phase consists of constructing a sequence by a dispatching rule which was controlled through parameters determined by the factors. In the third phase, a simulated annealing method was applied starting from a seed solution that resulted from their second phase.

Eom et al. (2002), proposed a three-phase heuristic to minimize the total weighted tardiness of a set of tasks with known processing times, due dates, weights and family types for parallel machines. In the first phase, jobs were listed by the earliest due date and then divided into small job-sets according to a decision parameter. In the second phase, jobs were grouped by the due date within applicable families using apparent tardiness cost with setup (ATCS), and the sequence of jobs within families is improved through the use of the tabu search (TS) method. In the third phase, jobs were allocated to machines using a threshold value and a look-ahead parameter.

Logendran and Subur (2004), reported a methodology for scheduling unrelated parallel machines with dynamic job releases and dynamic machine availability to minimize the total weighted tardiness. Four different methods based on simple and composite dispatching rules were used to identify an initial solution, which is then used by TS-based heuristic solution algorithm to ultimately find the best solution. The results showed that the newly developed composite dispatching heuristic, referred to as the apparent piecewise tardiness cost, is capable of obtaining initial solutions that significantly accelerate the TS-based heuristics to attain the best solution.

Mönch et al. (2005), proposed two different decomposition approaches to minimize total weighted tardiness on parallel batch machines with incompatible job families and unequal ready times of the jobs. The first approach forms fixed batches, then assigned these batches to the machines using a genetic algorithm (GA), and finally sequenced the batches on individual machines. The second approach assigned the jobs to machines first using a GA, then formed batches on each machine for the jobs assigned to it, and finally sequenced the batches. Dispatching rules were used for the batching phase and the sequencing phase of the two approaches.

Mönch (2008) presented an ant colony optimization (ACO) approach to solve unrelated parallel machine total weighted tardiness (TWT) scheduling problems. They used the Apparent Tardiness Cost (ATC) dispatching rule for the computation of the heuristic information.

There is an advantage in starting from a better initial solution than random solution in problems where good solutions cannot be easily obtained through a small number of

elementary transformations of a random solution (Pirlot, 1996). A good seed solution can decrease the computation time considerably (Lee and Pinedo, 1997).

3.3 Approximate Algorithms for the ARSP

In this dissertation, four algorithms APTCR, SA_{Random}, SA_{APTCR} and MetaRaPS were developed for ARSP to find near optimal solutions by taking into account the d-to-D window and jobs' release times. Firstly, a new composite dispatching rule, APTCR (Apparent piecewise tardiness cost with release time rule) is introduced which extends the ATC rule by modifying the priority index.

Secondly, simulated annealing (SA) has been successfully applied to different scheduling problems, and is therefore applied for the first time to the problem addressed in this research to obtain near-optimal solutions. SA that is a local search improvement heuristic, generates neighbor solutions by simple moves from the current solution starting with a randomly generated initial solution. The SA method was chosen as a robust meta-heuristic to provide immediate results to the problem. SA can deal with arbitrary systems and cost functions and is relatively easy to implement even for complex problems. It generally gives a good solution and statistically guarantees finding an optimal solution, but SA cannot tell whether it has found an optimal solution (Tan et al., 2001). Two general SA metaheuristic algorithms will be implemented to improve the random initial solution (SA_{Random}) and the initial solution constructed by APTCR (SA_{APTCR}). SA_{Random} is used as a way to evaluate the performance of the APTCR rule, whilst SA_{APTCR} results are compared to MetaRaPS especially for large problems.

Thirdly, MetaRaPS which is a new and promising randomized search metaheuristic, is applied to the problem by introducing randomness to the APTCR construction heuristic.

3.3.1 Apparent piecewise tardiness cost with release time (APTCR) Rule

In order to find a good initial solution, the Apparent Tardiness Cost (ATC), which is a good rule for the classical total tardiness scheduling problem, was modified while keeping its main rationale of calculating priority index for the jobs waiting to be scheduled. An ATC heuristic is a composite dispatching rule that combines the WSPT

(Weighted Shortest Processing Time) rule and the so-called Minimum Slack First (the job with the minimum slack is scheduled) rule (Pinedo, 2008). Several modifications of the ATC rule have been introduced in order to take release dates and sequence dependent setup times into account (e.g., Pfund et al., 2008). For the ARSP, a modified composite dispatching rule of Apparent Tardiness Cost rule has been developed taking d-to-D window into account to find scheduling priorities for the heuristic algorithm.

Generally ATC rules are based on the rationale that the highest priority job would be the one with the highest net savings for a unit of resource cost. Priorities may be estimated by the following formulation

$$\pi j = \frac{wj}{pj} Uj \qquad (10)$$

U_j is the marginal cost of delay; in other words activity time urgency function. The difficulty of this formula is the estimation of U_j (Morton and Pentico, 1993).

In the R&M heuristic (Rachamadugu and Morton, 1982), which is another version of ATC rule for weighted tardiness problems, priorities are calculated by rating the marginal benefit $B_j(t)$ of expediting the job (formulation (11)) to the marginal cost $C_j(t)$ of using the machine for that job (formulation (12)) (McKay et al., 2000).

$$B_j(\mathrm{t}) = wj \exp\left[-\frac{Sj+}{k\bar{p}}\right] \exp[-It] \qquad (11)$$

$$C_j(\mathrm{t}) = IRpj \exp[-It] \qquad (12)$$

t : Decision time point that the resource is considering which job to choose next.

R : The current implicit price of the machine,

I : The firm's (marginal) cost of capital.

$\bar{p}$: The average processing time of the unscheduled jobs at time t,

k : 'Look-ahead' or planning parameter and is set empirically.

S_j^+ (t) : a slack factor = $max\,[dj - pj - t, 0]$.

R and I can be assumed constant for all jobs. Then, the priority for a job (formulation (13)) is the ratio of benefit and cost.

$$\pi_j(t) = \frac{w_j}{p_j} \exp\left[- \frac{\max[d_j - p_j - t, 0]}{k_1 \bar{p}} \right] \tag{13}$$

Thus, the rule gives higher priorities to jobs approaching the due date. The due date for job j has no impact on the priority index when $C_j > d_j$.

The proposed APTCR heuristic is dynamic in a sense that after the completion of each job, the remaining jobs are prioritized according to priority index, and the job with the highest priority is selected for assigning to the earliest available machine. The priority index for job j at time t is calculated by (14)

$$\pi_j(t) = \frac{w_j}{p_j} \exp\left[- \frac{\max[d_j - p_j - t, 0]}{k_1 \bar{p}} \right] \times \exp\left[- \frac{\max[D_j - p_j - t, 0]}{k_2 \bar{p}} \right] \times \exp\left[- \frac{\max[r_j - t, 0]}{k_3 \bar{r}} \right] \times \frac{\max[D_j - p_j - t + \varepsilon, 0]}{[D_j - p_j - t + \varepsilon]} \tag{14}$$

where

$\bar{r}$ is the average ready time of the remaining jobs,

$max[D_j - p_j - t, 0]$ is deadline slack factor,

$max[r_j - t, 0]$ is ready time slack factor,

k_1, k_2, and k_3 are scaling parameters, called look-ahead parameter

$\max[D_j - p_j - t + \varepsilon, 0] / [D_j - p_j - t + \varepsilon]$ is scheduling control factor

The main rationale of this index is that getting closer to the due dates, deadlines, and ready times influence the priorities. The urgency of a job is measured by all these slack factors.

When the d-to-D window is considered, a new slack factor has to be included in the formulation. Because a much higher tardiness cost will be incurred if jobs miss the

deadline, a second slack factor may be defined as $max[Dj - pj - t, 0]$. Similar to due date, the deadline has an impact only before it has been reached. Completion time after the deadline results in a highest priority. The deadline factor is calculated by

$$\exp\left[-\frac{\max[D_j - p_j - t, 0]}{k_2 \bar{p}}\right] \tag{15}$$

Moreover, when the a job misses the deadline, its priority has to be assigned to zero after applying a high penalty because it will no longer belong to the pool of jobs to be scheduled. Then, in order to illustrate this situation, the scheduling-decision factor (16) has to be included as a multiplier.

$$\frac{\max[Dj - pj - t + \varepsilon, 0]}{[Dj - pj - t + \varepsilon]} \tag{16}$$

Before reaching the deadline for job j, the scheduling control factor helps identify a job that has missed its deadline but has no effect on the priority. A small number ε is added to allow assigning the job if the completion time happens to be exactly equal to the deadline.

Finally, the release time influences the priority index when it is larger than the current time, t. Thus, for dynamic release times, the factor (17) is included as a multiplier.

$$\exp\left[-\frac{\max[rj - t, 0]}{k3\bar{r}}\right] \tag{17}$$

The rationale here is that as the current time gets closer to the release date of job j, its priority increases, but after j is released (i.e. $r_j < t$), the urgency will vanish in this term and be reflected in the due date and deadline terms. The pseudo code for APTCR is given below in APTCR_ALGORITHM.

APTCR_ALGORITHM

t : decision time for assignment
C_j: completion time of job j

$Cmax_i\ (t)$: makespan of the machine i at time t
$\pi_j(t)$: priority of the job j at decision time t
U : set of undecided(to schedule or to unschedule) jobs
M : set of machines
n : number of jobs
r_j : release time of job j
p_j : processing time of job j
w_j : weight of job j
T_j : tardiness of job j
L: a large integer

1. Set $t=0$, $U=\{1,2,\ldots n\}$, $M=\{1,2,\ldots m\}$,
   $C_j = 0 \ \forall j \in U$, $Cmax_i(t) = 0 \ \forall i \in M$
2. Calculate $\pi_j(t) \ \forall j \in U$ by using equation (14)
3. **if** $\pi_j(t) = 0 \ \forall j \in U$ **then** remove j from U and set $C_j = L$
4. **while** $U \neq \emptyset$
5. Find $j = \{ j \in U : \pi j(t) = \max_{k \in U} \{\pi_k\} \}$
6. Find $i = \{ i \in M : Cmax_i(t) = \min_{m \in M} \{ Cmax_m(t) \} \}$
7. Update $C_j = Cmax_i(t) + max(r_j, t) + p_j$ and remove j from U
8. Update $Cmax_i(t) = C_j$
9. Update $t = \min_{m \in M} \{ Cmax_m(t) \}$
10. Calculate $\pi_j(t) \ \forall j \in U$ by using equation (14)
11. **if** $\pi_j(t) = 0 \ \forall j \in U$ **then** remove j from U and set $C_j = L$
12. **end while**
13. Calculate and report the total weighted tardiness

A sample for the problem with 12 jobs and 3 machines is given to clarify the methodology of obtaining the APTCR solution. Table 5 summarizes the priority index values calculated by using equation (14) for iterative time values. Every time the priority index is calculated, it takes into account the previous decisions as they affect the earliest starting time for the remaining jobs. The job with the highest priority value is assigned to the earliest available machine. According to Table 5, jobs 11, 2 and 3 which have the highest priority index values for t=0, are assigned to the three machines. After these assignments, the minimum of the latest completion times (or *Cmax*) is 25.5 at Machine 1. Job 7 is then assigned to follow Job 11 on Machine 1 as it has the maximum priority index value at t=25.5.

Jobs	M=1	M=2	M=3	M=1	M=2	M=3	M=1	M=2	M=1	M=3	M=2
	t=0	t=0	t=0	t=25 5	t=42 5	t=44	t=44 5	t=61 5	t=74 5	t=76	t=98 5
1	0 000009	0 000009	0 000009	0 02201	0 04526	0 04824	0 04927	0 08686	0 12067		
2	0 001625	0 001625									
3	0 000693	0 000693	0 000693								
4	0 000078	0 000078	0 000078	0 01035	0 02128	0 02268	0 02316	0 03946	0 05482	0 05694	0
5	0 000000	0 000000	0 000000	0 04748	0 09765						
6	0 000074	0 000074	0 000074	0 00979	0 01817	0 01887	0 01911	0 02937	0		
7	0 000000	0 000000	0 000000	0 05279							
8	0 000004	0 000004	0 000004	0 04664	0 09594	0 10224					
9	0 000295	0 000295	0 000295	0 03915	0 07267	0 07548	0 07644				
10	0 000074	0 000074	0 000074	0 03177	0 06535	0 06964	0 07113	0 11123			
11	0 019195										
12	0 000006	0 00001	0 00001	0 01025	0 02109	0 02247	0 02295	0 04082	0 05671	0 05890	

Table 5 APTCR Priority Index Values Calculated for Iterative Time Values

The scheduling using APTCR continues in a similar fashion until a complete solution is obtained as given in Table 6. Jobs whose completion times missed their deadline are denoted by very large completion times and are assigned to a dummy machine m+1 (machine 4 in this case). In Table 6, the first row is the job indices, the second row is the machine indices, and the third row is the completion times. For example, job 7 is scheduled on the machine 1 to complete at time 44.5. Job 4 and job 6 are not scheduled on any machines (zero priority index values in Table 5) and this situation is represented by scheduling them on the dummy machine 4 with very long completion time.

Job index	11	7	9	1	2	5	10	3	8	12	4	6
Machine index	1	1	1	1	2	2	2	3	3	3	4	4
Completion time	25.5	44 5	74.5	115.5	42.5	61.5	98 5	44	76	118	99999	99999

Table 6 Sample APTCR Solution

3.3.2 Simulated Annealing (SA)

Two general SA metaheuristic algorithms (SA_{Random} and SA_{APTCR}) will be implemented to improve the random initial solution and the initial solution constructed by APTCR. SA typically keeps one solution in memory and tries to move to improved solutions. The speed of a SA procedure depends on several parameters including the temperature cooling rate, the size of the neighborhood, and the nature of the objective function. The speed as well as the quality of the SA is also dependent on how fast the algorithm can find a good neighbor, which may lead to an optimal or near-optimal solution. Selecting the most promising candidates in the search for a good neighbor accelerates the convergence towards a high quality solution. Pseudo code for SA is given below.

SA_ALGORITHM

S: search space

θ *:* current solution

$f(\theta)$ *:* objective function value of the current solution

Memory: memory set of current best solution and its objective function value

i *:* inner loop iteration counter

i_{max} *:* max number of inner loop iterations

t *:* iteration counter

t_{max} *:* max number of iterations

T: temperature

k *:* initial temperature coefficient

α *:* temperature cooling coefficient

$N(\theta)$ *:* neighborhood of θ

θ' *:* neighbor solution of the current solution

θ^* *:* best solution

1. Get random or APTCR solution as θ from S using *APTCR_ALGORITHM*
2. Calculate $f(\theta)$

3. Initialize memory, $Memory = \{(\theta, f(\theta))\}$
4. Set iteration counters $i = 0$, $t = 0$
5. Set initial temperature $T = k.f(\theta)$
6. **while** $t < t_{MAX}$ **do**
7. **while** $i < i_{MAX}$ **do**
8. Choose $\theta' \in N(\theta) \subseteq S$ *where* $M = \{(\theta, f(\theta))\}$
9. Calculate $f(\theta')$
10. **if** $f(\theta') \leq f(\theta)$ **or** rand $[0,1] \leq e^{-\frac{f(\theta')-f(\theta)}{T}}$ **then**
11. $M = \{(\theta', f(\theta'))\}$
12. **end if**
13 $i = i+1$
14. **end while**
15. Update temperature $T = \alpha.T$
16. $i = 0$, $t = t+1$
17. **end while**
18. Output best solution θ^* stored in M

SA has four parameters: *maximum number of outer loop iterations(t_{max}), maximum number of inner loop iterations(i_{max}), initial temperature coefficient(k)* and *temperature cooling coefficient(α)* and these parameter will be tuned in Section 4.3.2 for better solutions.

The mixture of two types of operations (job exchange and job insertion) is used for the SA to perturb the current solution locally. In order to avoid high unavailability costs when there are unscheduled jobs, job insertion is executed by inserting one of the unscheduled jobs to replace a scheduled job and shifting the jobs to the right on the corresponding machine. In the job insertion case, the size of the neighborhood is $n-1+m$ where n is the number jobs and m is number of machines. If there are no unscheduled jobs, then job exchange is used for perturbation in which case the size of the

neighborhood would be *n.(n-1)* possibilities. After each perturbation, completion times on the corresponding machines are revised according to the new sequence.

The SA perturbation operations are explained in Table 7 for the sample APTCR solution given in Table 6. There are two alternatives for perturbation. First, if there does not exist an unscheduled job, an exchange between randomly selected job 10 (the third job on Machine 2) and job 11 (the first job on Machine 1) is performed for perturbing. As can be seen in Table 7, job 1 and job 11 eventually miss their deadlines when completion times on the Machine 1 and Machine 2 are revised. Second, if there exists an unscheduled job (e.g. job 6), job insertion is performed by inserting job 6 into the left of a randomly selected job (e.g. job 1). In this case, only job 1 on the right of the newly inserted job 6 is shifted causing it to miss its deadline as can be seen in Table 8.

Job Exchange	10	7	9	1	2	5	11	3	8	12	4	6
Jobs 10 and 11	1	1	1	4	2	2	4	3	3	3	4	4
	45.5	64.5	94.5	99999	42.5	61.5	99999	44	76	118	99999	99999

Table 7 Job Exchange Operation for SA Perturbation

Job Insertion	11	7	9	6	1	2	5	10	3	8	12	4
Job 6 before Job 1	1	1	1	1	4	2	2	2	3	3	3	4
	25.5	44.5	74.5	118.5	99999	42.5	61.5	98.5	44	76	118	99999

Table 8 Job Insertion Operation for SA Perturbation

3.3.3 Metaheuristic for Randomized Priority Search (MetaRaPS)

The construction phase in MetaRaPS algorithm builds a solution by systematically adding feasible jobs to the current schedule on identical parallel machines. This study will consider the use of randomized version of APTCR composite dispatching rule to construct initial solutions which will be improved by a local search operation. The pseudo code for an iteration of MetaRaPS algorithm is given below.

MetaRaPS_ALGORITHM

θ : constructed feasible solution

f(0) : objective function value of the constructed solution

1. Set t=0, $U = \{1,2,\dots n\}, M = \{1,2,\dots m\}$, $Memory = \{(fbest, fworst)\}$
2. $Cj = 0 \ \forall j \in U$, $\text{Cmax}_i(t) = 0 \ \forall i \in M$, CL = ∅
3. Calculate $\pi_j(t) \ \forall j \in U$ by using equation (11)
4. **if** $\pi_j(t) = 0 \ \forall j \in U$ **then** remove j from U and set C_j =a large integer
5. **while** U ≠ ∅
6. Find i = {i ∈ M : $\text{Cmax}_i(t) = \min_{m \in M}\{\text{Cmax}_m(t)\}$ }
7. Find j = { $j \in U : \pi j(t) = \max_{k \in U} \{\pi_k\}$ }
8. P = RND(0,1)
9. **if** P ≤ %p **then**

   Set job-to-schedule index, s = j
10. **else**

    Form CL = {k : k ∈ U $|\pi_k(t) \geq \pi_j(t)$. %r }

    Randomly choose job k from CL

    Set job-to-schedule index, s = k and CL = ∅
11. **end if**
12. Update $C_s = \text{Cmax}_i(t) + \max(r_s, t) + p_s$ and remove job s from U
13. Update $\text{Cmax}_i(t) = C_s$
14. Update t = $\min_{m \in M}\{\text{Cmax}_m(t)\}$

15. Calculate $\pi_j(t)\ \forall j \in U$ by using equation (11)
16. **if** $\pi_j(t) = 0\ \forall j \in U$ **then** remove j from U and set C_j =a large integer
17. **end while**
18. Calculate f(θ)
19. **if** $f(\theta) \leq f_{best} + (f_{worst} - f_{best}).\%i$ **then**
20. Find $\theta' \in N(\theta)$:

    if at least one unscheduled job exists, use job insertion,

    else use job exchange
21. Calculate $f(\theta')$
22. **end if**
23. Update Memory

The APTCR index for each job is calculated as discussed in Section 3.3.1, and the selection is made based on MetaRaPS principles. If the random number is smaller or equal to the priority percentage, the job with the highest APTCR priority index ($\pi_j(t)$) is selected. The remaining time, jobs whose priority indices are higher than the lower limit ($\pi_j(t).\ \%r$) are added to the candidate list (CL) and the next job is selected from this CL randomly. After all jobs are assigned to machines, the construction phase of MetaRaPS is completed. A sample data given in Table 9 for problem with m = 3 machines and n = 12 jobs, was randomly generated to explain MetaRaPS algorithm for ARSP.

The construction phase of MetaRaPS algorithm iteration is shown in Table 10. In every step of this phase, jobs are assigned to the machines with the earliest availability. In the first step, job 10 has the maximum priority index value. Jobs 2, 4 and 5 are in the CL because their priority values are higher than 0.0240 (=0.0959 x 0.25). Since the generated random number (RN) is less than %p(0.21 < 0.75), job 10 is selected to be assigned to Machine 1 without using the CL.

Job	Release Time	Processing Time	Weight	Due Date	Deadline
1	19	30	4	79	109
2	1	28	7	55	83
3	24	23	8	70	103
4	1	28	8	55	83
5	3	19	6	41	60
6	10	45	5	100	145
7	2	45	4	92	137
8	5	43	3	91	134
9	11	28	1	67	95
10	0	23	8	46	69
11	13	29	1	71	100
12	15	32	2	79	111

Table 9 Sample Data for ARSP

Step	Machine 1	Machine 2	Machine 3	Max. Priority Index Value	Job with Max. Priority	Lower Limit r=0.25	CL	RN	RN > p = 0.75	Job Selection Criteria	Scheduled Job	Machine
1	0	0	0	0 0959	10	0 0240	2,4,5	0 21	no	Max Priority	10	1
2	23	0	0	0 0800	5	0 0200	2,4	0 85	yes	From CL	2	2
3	23	28	0	0 0800	5	0 0200	4	0 77	yes	From CL	4	3
4	23	28	28	0 2167	5	0 0542	3	0 56	yes	Max Priority	5	1
5	42	28	28	0 0895	3	0 0224	1	0 37	yes	Max Priority	3	2
6	42	51	28	0 0340	1	0 0085	6,7,8,9,11,12	0 81	yes	From CL	8	3
7	42	51	71	0 0557	1	0 0139	6,7,9,11,12	0 21	no	Max Priority	1	1
8	72	51	71	0 0377	7	0 0094	6,9 11,12	0 79	yes	From CL	6	2
9	72	96	71	0 0573	7	0 0143	12	0 93	yes	From CL	12	3
10	117	96	103	0.0585	7	-	-	-	-		7	1
										Step 9	9 11	None

Table 10 Construction Phase of MetaRaPS Algorithm

The latest completion time of machine 1 is updated as the completion time of job 10. The minimum latest completion time (t = 0) does not change. Job 5 is the highest priority job and jobs 2 and 4 are in the CL in the second step. Differently from the first step, RN> %p (0.85 > 0.75) and therefore job 2 is randomly selected from CL to be assigned to Machine 2. Finally, jobs 9 and 11 are not assigned to any machine as their completion times will exceed their deadlines.

Since the best feasible job is the job with the highest priority index, the improvement phase is performed when the constructed feasible solution value is less than the limit value. After a number of iterations, MetaRaPS tracks the best and worst solutions found during the construction phase. A constructed solution goes through the improvement phase by the same perturbation used with SA (see Section 3.3.2 for the details of job exchange and insertion operations) if its objective function value satisfies the following requirement;

$$f(\theta) \leq f_{best} + (f_{worst} - f_{best}).\%i$$

where $f(\theta)$ is the objective function value of the constructed solution, f_{best} is the best and f_{worst} is the worst objective function value found before applying the improvement procedure.

3.4 Complexity Analysis

Complexity of an algorithm and complexity of a problem which are related but different concepts are analyzed for ARSP and its algorithms. Firstly, the complexity of an algorithm for a certain problem is measured by the maximum (worst-case) number of computational steps needed to obtain an optimal solution as a function of the size of the instance (i.e., the number of jobs) (Pinedo, 2008). The number of computational steps may often be the approximated maximum number of iterations of the algorithm. In order to be independent of a particular type of a computer the deterministic Turing machine *(DTM)* is used as an abstract model of computation (Blazewicz et al., 2007). *Worst case running time* is the behavior of the algorithm with respect to the worst possible case of the input instance. It is an upper bound on the running time for any input. An algorithm whose order-of-magnitude time performance is bounded by a polynomial function of n,

where n is the size of its inputs, is called a *polynomial-time algorithm.* It is one whose time complexity function is *O(p(k))*, where p is some polynomial and k is the input length of an instance. Each algorithm whose time complexity function cannot be bounded in that way will be called an *exponential time algorithm.* In practice, the size of an instance is often simply characterized by the number of jobs (n). For example, if the maximum number of iterations needed to obtain an optimal solution is $1200 + 50n^2 + 4n^3$, then only the term which, as a function of n, increases the fastest is of importance. This algorithm is then referred to as an $O(n^3)$ algorithm. An $O(n^3)$ algorithm is usually referred to as a polynomial time algorithm; the number of iterations is polynomial in the size (n) of the problem. As another example, a simple merge sort can be done in *O(n log(n))* time.

As for problem complexity, a problem that admits a reasonable or polynomial-time solution is said to be *tractable*, whereas a problem that admits only unreasonable or exponential-time solutions is termed *intractable.* In general, intractable problems require impractically large amounts of time even on relatively small inputs, whereas tractable problems admit algorithms that are practical for reasonably-sized inputs (Harel and Feldman, 2004). A significant amount of research in deterministic scheduling has been devoted to finding polynomial time algorithms for scheduling problems. However, many scheduling problems do not have a polynomial time algorithm; these problems are the so-called *NP-hard* problems. NP stands for the class of problems that admit nondeterministic polynomial- time algorithms, P stands for what we have been calling tractable problems; namely, those that admit polynomial-time algorithms. This would mean that P is a proper subclass of *NP* and the classes *P* and *NP-complete* problems are disjoint. The NP-complete problems, are the "hardest" problems in NP, in the sense that there are polynomial-time reductions from every problem in NP to each of them.

A fundamental concept in complexity theory is the concept of *problem reduction.* Very often it occurs that one combinatorial problem is a special case of another problem, or equivalent to another problem, or more general than another problem. Often, an algorithm for one scheduling problem can be applied to another scheduling problem as well. For example, the problem $1|\ |\Sigma w_j T_j$ is an important generalization of the $1|\ |\Sigma T_j$ problem. In complexity terminology it is then said that $1|\ |\Sigma T_j$ *reduces* to $1|\ |\Sigma w_j T_j$, which

is an *NP-hard* problem (Pinedo, 2008). This means that no efficient algorithm can be obtained for $1|\ |\Sigma w_j T_j$ with arbitrary weights. Moreover, the identical parallel machine scheduling is NP-hard even with only two machines (Karp, 1972; Garey and Johnson, 1979). Thus, ARSP with release time and d-to-D window must also be NP-hard, which means that obtaining optimal solutions for large instances will be computationally difficult. There are n+1 different cases of the assignment and permutation of n jobs to m machines including the possibilities of jobs being unscheduled. In Case 1 below for example, all n jobs are scheduled for which the number of possible combinations is as shown. Case 2, however, captures the case of scheduling n-1 jobs with 1 job left unscheduled. It is important to note that the number of possibilities includes permutations only for jobs that have been scheduled as the sequence (or permutation) of the unscheduled jobs is not important.

Case	Number of Scheduled Jobs	Possibilities
1	n	$\binom{n}{n}.n!.m^n$
2	n-1	$\binom{n}{n-1}.(n-1)!.m^{n-1}$
3	n-2	$\binom{n}{n-2}.(n-2)!.m^{n-2}$
.		
.		
n	1	$\binom{n}{1}.1!.m$
n+1	0	$\binom{n}{0}$

Jobs can be selected for scheduling according to k combination of n which is defined as

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

For instance $n-1$ case captures the number of possibilities in which n-1 can be scheduled to machines assuming only one job is left unscheduled. There are $n!m$ possibilities to schedule n jobs on m machines because there is a total of m possibilities for machine assignment, and $n!$ permutations on each machine. Total schedule possibilities including unscheduled jobs is sum of all possibilities for all n cases:

$$\binom{n}{n}.n!.m^n + \binom{n}{n-1}.(n-1)!.m^{n-1} + \ldots + \binom{n}{1}.1!.m + \binom{n}{0}$$

For example, when it is assumed that n=4 and m=2, the total number of possible combination is as follows:

$$\binom{4}{4}.4!.2^4 + \binom{4}{3}.3!.2^3 + \binom{4}{2}.2!.2^2 + \binom{4}{1}.1!.2 + 1 = 637$$

According to worst case running time analysis in Table 11, all three algorithms have $O(n^2)$ order of complexity.

Algorithm	Worst case Running Time	Algorithm Order
APTCR	For each n assignment: Calculate the index values O(n), Find the job with max index O(n), Find the machine with min completion time O(m), Check deadline and Calculate TWT O(n), n>m	$O(n^2)$
SA	For each iteration (tmax): Generate the initial solution O (n), Improvement $O(n^2)$ (swapping n.(n-1) or insertion (n+m-1)) Check deadline and Calculate TWT O(n). Comparison O(1)	$O(n^2)$
MetaRaPS	For each iteration (tmax): Construction of n jobs: Calculate the index values O(n), Find max and min index O(n), Select one of best r jobs O(n), Find the machine with min completion time O(m) Check deadline and Calculate TWT O(n), Improvement $O(n^2)$	$O(n^2)$

Table 11 Complexity Analysis of Scheduling Algorithms

CHAPTER 4

COMPUTATIONAL STUDY FOR ARSP'S SCHEDULING ALGORITHMS

Computational study for the ARSP algorithms contains estimating the look-ahead parameters values for APTCR, parameter setting for SA (same parameters were used for both SA versions) and MetaRaPS, and determining the performance of the proposed algorithms for small (n=12) and large (n=60) size problems. In order to perform a fair comparison, APTCR - SA_{random} pair and SA_{APTCR} - MetaRaPS pair which are used to improve an initial solution constructed by APTCR rule, were compared separately. For all of these study, some extensive experiments were designed and conducted.

Design of Experiments (DOE) is a useful systematic approach for resolving multidimensional problems such as determining the relationship between input factors and process outputs. A primary goal of many DOEs is to identify which of the factors are really important for which responses, and which are not and can thus be dropped from further consideration, greatly reducing the experimental effort and simplifying the task of interpreting the results (Sanchez, 2008). An experimental design formally represents a sequence of experiments to be performed, expressed in terms of *factors* (design variables) set at specified *levels* (predefined values). An experimental design is represented mathematically by a matrix where the rows denote experimental runs and the columns denote the particular factor setting for each factor for each run (Simpson et al., 1997). The collected response data is often applied to fit mathematical equations that serve as models to predict the outcome with any given combination of values. One of the steps in a typical parameter design study is designing the matrix experiment after identifying the quality characteristic to be observed and the objective function to be optimized, identifying the design parameters and alternative levels, and defining possible interactions between these parameters (Phadke, 1989). Many designs are available in the literature such as full factorial design, central composite design, orthogonal array design, D-optimal design, and signal-to-noise (S/N) ratio method.

Multi factorial design with experiment factors and factor levels was employed and their high level interactions were analyzed to gain insight into the algorithms' performance under different conditions (See Montgomery (2001) for discussion on DOE). The most basic experimental design for multi levels is a *full factorial design*. Factorial designs are straightforward to construct and readily explainable—even to those without statistical backgrounds. They examine all possible combinations of the factor levels for each of the variables (Sanchez, 2008). The number of design points dictated by a full factorial design is the product of the number of levels for each factor. The most common designs are the 2^n (for evaluating main effects and interactions) and 3^n designs (for evaluating main and quadratic effects and interactions) for n factors at 2 and 3 levels, respectively. 2^n factorials are most efficient if the simulation response is assumed that it is well-fit by a model with only linear main effects and interactions, while 3^n factorials provide greater detail about the response and greater flexibility for constructing metamodels of the responses. Multilevel full factorial design can reveal complexities in the landscape. When each factor has three levels, the convention is to use -1, 0, and 1 for the coded levels. Despite the greater detail provided, and the ease of interpreting the results, fine grids are not suitable for more than a handful of factors because of their massive data requirements (Sanchez, 2008).

The proposed approximate algorithms were implemented in C++. Optimal solutions were obtained by implementing the MILP model (Section 3.1) in OPL Studio 6.3 with CPLEX 12.1 solver. Intel Core 2 Duo 2.10 GHz CPU with 2.00 GB of RAM was used to perform the computations. Note that the MILP model could be used only for instances with 12 jobs in order to limit the CPU time to less than 10 minutes.

4.1 Experiment Factors

The design of experiments approach to estimate the look ahead parameters is described below. Four different factors are included in the design: *Job machine factor (μ), Due date tightness (α), d-to-D window tightness (γ)* and *Release time range factor (ρ)* which characterize a problem instance as follows :

1. *Job machine factor* : $\mu = n/m$, the average number of jobs processed per machine.

2. *Due date tightness* factor α : It characterizes how tight the due dates are and is defined as a coefficient in $d_j = r_j + \alpha.p_j$ by taking into account fuel level and refueling time. The due date tightness α is assessed by the decision maker, and should rationally take values greater than 1 to provide enough time to complete the process. A small value of α indicates tight due dates and a large α indicates loose due dates.

3. *Due date-to-Deadline (d-to-D) window tightness* factor γ : It characterizes how tight the d-to-D windows are. The deadline can be defined as $D_j = d_j + \gamma.p_j$. Consequently, the d-to-D window is calculated by $D_j - d_j = \gamma.p_j$. The d-to-D window factor γ is assumed always higher than zero. A small value of γ indicates tight d-to-D windows and a large γ loose d-to-D windows.

4. *Release time range factor* ρ is a measure of how spread out the release times are as compared to the estimated makespan ($\hat{C}_{max}$). The makespan (C_{max}) is the maximum completion time of all released jobs. Since C_{max} is dependent on the schedule and is not known apriori, an estimated makespan ($\hat{C}_{max}$) will be developed in the following subsection.

4.2 Data Generation

1. The number of jobs is set to a low value (n = 12) and a high value (n = 60) and then the number of machines is determined by using $m = n/\mu$ where μ is the job machine factor.

2. The processing times p_j are uniformly distributed integers in the interval [15, 45].

3. The weights for the jobs w_j are uniformly distributed integers in the interval [1, 9].

4. Given the average of jobs' processing times ($\bar{p}$), the average of the jobs' release times ($\bar{r}$), job machine factor (μ), and coefficient (ϕ), which takes into account the effect of the release times on the makespan (assumed here $\phi = 0.1$), then $\hat{C}_{max} = (\phi\bar{r} + \bar{p}).\mu$ estimates C_{max} of the problem. A similar approach was used by Lee and Pinedo (1997).

5. Release times r_j are uniformly distributed in the interval [0, $2\bar{r}$] where $\bar{r}$ is the jobs' release time average. The maximum release time is derived as $2\bar{r} = 2\bar{p}\mu\rho/(2 - \phi\mu\rho)$ where

the release time range factor (ρ) is $\rho = 2\bar{r} / \hat{C}_{max}$ and the estimated makespan is $\hat{C}_{max} = (\phi\bar{r} + \bar{p}).\mu$ as defined earlier.

6. Due dates are calculated by $d_j = r_j + \alpha.p_j$ formula and deadlines are calculated by $D_j = d_j + \gamma.p_j$ formula.

4.3 Parameter Setting

A parameter setting procedure can be dynamic (online) or static (offline). Dynamic parameter setting procedures merge the parameter setting and solution building phases for a heuristic. Dynamic parameter setting methods sample different parameter setting levels and then converge on the "best found" parameter setting level and ultimately report the best solution found by the heuristic.

In this study, parameters of the algorithms were tuned by extensive experiments to obtain better results. Multi factorial design with four factors and three levels was employed in parameter tuning because all high level interactions are desired to analyze by using any possible combination of the instance factors and cost of each experiment is not too high (Montgomery, 2001).

4.3.1 APTCR Parameter Setting

The values of the parameters which make the proposed APTCR algorithm work effectively were determined through extensive experiments. The look-ahead parameters are dependent on the particular problem instance in terms of job machine factor (μ), due date tightness (α), d-to-D window tightness (γ) and release time range factor (ρ) factors. Thus, an experimental study was conducted to determine the values of $k_1 = f_1(\mu, \alpha, \gamma, \rho)$, $k_2 = f_2(\mu, \alpha, \gamma, \rho)$ and $k_3 = f_3(\mu, \alpha, \gamma, \rho)$ in (17). Look-ahead parameters (k_1, k_2 and k_3) are additional factors of the experiment besides other experiment factors. The parameter values resulting in the minimum total weighted tardiness values for each problem instance are the response values of the experiment. Regression equations mapping the factors of an instance into values of the three look-ahead parameters were determined to estimate the parameters.

Extensive experiments were conducted over three levels of the job (n = 60) machine factor (μ=4, μ=5, μ=6), three levels of the due date tightness factor (α =1.5, α =2.5, α =3.5), three levels of the d-to-D window tightness factor (γ = 0, γ = 1, γ = 2) and three levels of the release time range factor (ρ = 0.1, ρ = 0.2, ρ = 0.3).

81 (3^4) unique problem instances were generated according to specific distributions for each input data and replications. Within each problem type, five problem instances were generated by using distribution functions, totaling 405 problem instances. 32 levels were used for each look ahead parameters as below.

32 levels of k_1: {0.2, 0.4,....., 6.4}

32 levels of k_2: {0.2, 0.4,....., 6.4}

32 levels of k_3: {0.2, 0.4,....., 6.4}

For each of the 405 instances, the total weighted tardiness values of 32,768 (32x32x32) combinations of k_1 , k_2 and k_3 were evaluated. In order to compare the main factor and interaction effects with each other, coded factors values, given in Table 12, were used to find the mathematical model of the response value.

Level	Coded	Actual			
		μ	α	γ	ρ
Low	-1	4	1.5	0	0.1
Medium	0	5	2.5	1	0.2
High	1	6	3.5	2	0.3

Table 12 Coded Values of Factor Levels

All total weighted tardiness that were obtained applying the algorithm repeatedly for 32,768 combinations of k_1 , k_2 and k_3 were evaluated. All k_1 , k_2 and k_3 values that yielded in the range between the minimum total weighted tardiness (MTWT) and MTWT(1 + β) for each instance, were identified. The averages of these k_1 , k_2 and k_3 values are denoted

as $\bar{k}_1$, $\bar{k}_2$ and $\bar{k}_3$. The parameter β is a tolerance value for MTWT that is a decreasing function of due date tightness factor and ranges from 0 to 0.065 (Lee and Pinedo, 1997). In this study, the highest value, 0.065 of this tolerance parameter was used without calculation to evaluate a large range of competing k values. The averages of the selected range of k_1, k_2, and k_3 are used as the recommended values for k_1, k_2, and k_3. Thus, each problem instance has a triple of recommended values for k_1 , k_2, and k_3. A C++ program was used to generate instances and to find schedules, TWT values, and recommended k values. Then averages of the instance recommended values, as given in Appendix B, were calculated to use in the regression analysis.

In order to find the best fit model, different kinds of transformations for response values such as logarithmic, square root, square, reciprocal, and exponential were attempted. Then as a result of the regression analysis using realized values of the problem instance characteristics as regressor and the scaling parameters as a response, regression equations of the look-ahead parameters were obtained.

$$k_1^2 = 4.514 + 0.693\mu - 1.626\alpha - 0.921\gamma - 1.499\rho + 1.064\mu^2 - 1.659\gamma^2 - 0.668\mu\gamma - 0.670\mu\rho + 1.069\alpha\rho$$

$$k_2 = 0.980 + 0.275\gamma + 0.438\mu^2 + 1.02\,\gamma^2 + 0.304\gamma\rho$$

$$k_3 = (0.437 - 0.152\mu - 0.090\alpha - 0.144\gamma + 0.598\alpha^2 + 0.226\gamma^2 - 0.359\mu\alpha + 0.083\alpha\gamma + 0.100\alpha\rho)^2$$

The best models for the k_1, k_2, and k_3 have R^2 values of 0.59, 0.50, and 0.78 respectively.

4.3.2 SA Parameter Setting

Same parameters were used for both SA_{random} and SA_{APTCR}. Experiments using the standard setting of 3 machines and 12 jobs has been conducted to obtain appropriate parameter values for the SA algorithm over three levels of maximum number of iterations (t_{max}=1000, 2000, 4000), three levels of maximum number of inner loop iterations (i_{max} = 5, 10, 15), three levels of initial temperature coefficient (k = 0.1, 1, 10) and three levels of temperature cooling coefficient (α=0.7, 0.8, 0.9). For each 81 (3^4) combinations of

parameters, 5 problem instances were solved 30 times by SA starting from random initial solutions.

After performing regression analysis of the total weighted tardiness values, the following parameter levels were determined: initial temperature coefficient = 0.1, temperature cooling coefficient = 0.7, maximum number of inner loop iterations = 5, and maximum number of iterations = 4000.

4.3.3 MetaRaPS Parameter Setting

The solution quality of the MetaRaPS depends on *I*, *%p*, *%r* and *%I* parameters. Experiments using the standard setting of 3 machines and 12 jobs has been conducted to determine the values of parameters for the MetaRaPS algorithm over nine levels of *%p* (*0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9*), nine levels of *%r* (*0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9*), five levels of *%i* (0.4, 0.5, 0.6, 0.7, 0.8) with increment sizes of 10%. High values of *I* and %i parameters are always preferred and the value of these parameters depends on the availability of computation time (Hepdogan et al., 2009). The number of iterations, *I*, was fixed to 2000 which results in compatetive CPU times with SA_{APTCR} algorithm.

For each 405 (9x9x5) combination of parameters, 5 problem instances generated randomly for each medium level of four problem factors, were solved 10 times by MetaRaPS. Average relative error of replicative solutions from optimal solution was calculated for each parameter combination by using the equation (18). After performing regression analysis of the average relative error values, the following parameter levels shown in Table 13 were determined.

4.4 Performance of the Scheduling Algorithms

The efficacy of the proposed approximate algorithm was tested by the quality of schedule generated by algorithm. Two performance measures were considered for the proposed algorithm: *quality of the best solution* obtained and the *computation time* it takes. To measure the effectiveness of the algorithms, they were compared for low and high (n=60) levels of number of jobs.

Parameter	Value
Number of iterations (I)	2000
Priority percentage (p%)	75%
Restriction percentage (r%)	25%
Improvement percentage (i%)	75%

Table 13 MetaRaPS Parameter Setting

4.4.1 Effectiveness of APTCR-SA_{random} for Small Size Problems

The MILP model was used for solving small size problems only with up to 12 jobs in acceptable time. Thus, the performance of the algorithms are measured for these problems by computing the difference between the objective function values (total weighted tardiness) of the algorithm and the optimal schedules as follows:

$$\text{Relative Error} = \frac{TWT_{Algorithm} - TWT_{Optimal}}{TWT_{Optimal}} \quad (18)$$

The performance of the APTCR and SA_{random} algorithms were compared in terms of relative error and computation time over low and high levels of the problem factors. 10 different unique problem instances were generated for each 16 (2^4) factor combinations. SA_{random} solutions were replicated 10 times for each of the 160 instances and the average, minimum and maximum relative errors were recorded. Average relative errors (APTCR, SA_{random}, Min. SA_{random} and Max. SA_{random}) and average CPU times are found by averaging values of the 10 instances for each combination. Table 14 summarizes the results.

μ	α	γ	ρ	Avg. Relative Error		Min. Relative Error	Max. Relative Error	Avg. CPU (sec.)	
				APTCR	SA_{random}	SA_{random}	SA_{random}	APTCR	SA_{random}
3	1.5	0	0.1	0.14	0.04	0.01	0.10	0.00	0.07
			0.3	0.24	0.09	0.00	0.19	0.00	0.08
		2	0.1	**0.30**	0.46	0.14	0.77	0.00	0.11
			0.3	**0.23**	1.01	0.40	1.64	0.00	0.11
	2	0	0.1	1.33	0.28	0.00	0.58	0.00	0.09
			0.3	1.40	0.30	0.27	0.38	0.00	0.09
		2	0.1	**0.52**	5.67	3.09	8.20	0.00	0.10
			0.3	**1.91**	7.21	3.55	11.11	0.00	0.09
6	1.5	0	0.1	0.06	0.00	0.00	0.01	0.00	0.07
			0.3	0.16	0.05	0.00	0.09	0.00	0.07
		2	0.1	0.11	0.04	0.01	0.08	0.00	0.10
			0.3	0.47	0.15	0.01	0.23	0.00	0.09
	2	0	0.1	0.12	0.02	0.00	0.03	0.00	0.07
			0.3	0.28	0.17	0.02	0.28	0.00	0.08
		2	0.1	0.21	0.15	0.03	0.27	0.00	0.10
			0.3	0.62	0.34	0.02	0.62	0.00	0.09
Total Average				**0.51**	**1.00**	0.47	1.54	0.00	0.09

Table 14 Results of the APTCR and SA_{random} for problems with n= 12

The results show that SA_{random} has performed better than APTCR in terms of average relative error for most instances; however, the proposed heuristic algorithm APTCR was superior to SA_{random} for loose d-to-D windows with small job-machine factor. The average CPU times of SA_{random} were less than 1 second for each problem instance, but it is obvious that they are much longer (~60 times on the average) than APTCR.

Since the relative error data for each combination does not follow a normal distribution according to Anderson-Darling normality test, the statistical significance of performance between algorithms are analyzed via using nonparametric Kruskal–Wallis test (using Minitab 15.1 statistical software program). Table 15 shows that there is a statistical difference between the population mean values of APTCR and SA_{random} performance in

terms of relative error ($p = 0.007 < \alpha=0.05$). SA_{random} has performed better because the median of SA_{random} relative errors is less than the median value of APTCR ($0.097 < 0.203$). On the other hand, APTCR has better CPU time performance than SA_{random} by having significantly smaller median value.

Kruskal-Wallis Test on Relative Error

Method	N	Median	Ave Rank	Z
APTCR	160	0.20266	174.4	2.68
SA_{random}	160	0.09735	146.6	-2.68
Overall	320		160.5	

H = 7.18 DF = 1 P = 0.007
H = 7.21 DF = 1 P = 0.007 (adjusted for ties)

Kruskal-Wallis Test on CPU

Method	N	Median	Ave Rank	Z
APTCR	160	0.001000	80.5	-15.47
SA_{random}	160	0.091750	240.5	15.47
Overall	320		160.5	

H = 239.25 DF = 1 P = 0.000
H = 241.19 DF = 1 P = 0.000 (adjusted for ties)

Table 15 Comparison of Effectiveness of the Algorithms for n= 12 Problems

If the results are analyzed separately for APTCR and SA_{random} by Kruskal-Wallis test, the average relative error effectiveness of both algorithms decreases significantly as the number of jobs per machine decreases (from 6 to 3), the due date tightness decreases (α increases from 1.5 to 2), and the release time tightness increases (ρ increases from 0.1 to 0.3). When the d-to-D window tightness decreases (γ increases from 0 to 2) only SA_{random} effectiveness decreases significantly. The due date tightness factor has the highest effect on the APTCR performance, while the d-to-D window tightness factor and job-machine factor have the highest effect on the SA_{random} performance. The average CPU time effectiveness of both algorithms does not change significantly between the problem factor levels.

Levene's test which is suitable when the data come from continuous, but not necessarily normal distributions was used to test the equality of variance which is another important measure of robustness of the algorithm solutions. The test shows that there exists significant difference between variances for both the relative error ($p = 0.045 < \alpha= 0.05$)

and CPU time ($p = 0.000 < \alpha = 0.05$) (Table 16) indicating that it is more robust for this problem compared to SA_{random}.

Test for Equal Variances: Relative Error	Test for Equal Variances: CPU
95% Bonferroni confidence intervals for standard deviations	95% Bonferroni confidence intervals for standard deviations
Method N Lower St Dev Upper APTCR 160 0.75518 0.85040 0.97188 SA_{random} 160 2.93976 3.31041 3.78328	Method N Lower St Dev Upper APTCR 160 0.0026969 0.0030370 0.0034708 SA_{random} 160 0.0181035 0.0203861 0.0232981
Levene's Test (Any Continuous Distribution) Test statistic = 4.04; p-value = 0.045	Levene's Test (Any Continuous Distribution) Test statistic = 231.23; p-value = 0.000

Table 16 Test for Equal Variances: Relative Error and CPU versus Method

4.4.2 Effectiveness of APTCR- SA_{random} for Large Size Problems

The same APTCR parameter estimations and parameter values are used to test the effectiveness for large size problems with 60 jobs. Since no optimal solutions exist for these problem sizes, the quality of the solution was measured by relative difference from the best of both algorithms to avoid nonnegative performance values.

$$\text{Relative Difference} = \frac{TWT_{Algorithm} - min(TWT_{MATC}, TWT_{SA\ random})}{min(TWT_{MATC}, TWT_{SA\ random})} \quad (19)$$

The relative difference was calculated for each of the 160 problem instances then the averages were used for each problem instance combination. In (19), $TWT_{SA\text{-}random}$ is the average of SA_{random} solutions for 10 replications. The comparison results for large size problems in terms of average relative difference and average CPU time are given in Table 17.

μ	α	γ	P	Avg. Relative		Avg. CPU (sec.)	
				APTCR	SA_{random}	APTCR	SA_{random}
3	1.5	0	0.1	0.00	0.63	0.00	0.48
			0.3	0.00	1.00	0.01	0.50
		2	0.1	0.00	0.98	0.01	0.98
			0.3	0.00	1.50	0.01	0.98
	2	0	0.1	0.00	1.75	0.01	0.59
			0.3	0.00	1.72	0.01	0.61
		2	0.1	0.00	3.20	0.01	0.99
			0.3	0.00	7.83	0.01	0.99
6	1.5	0	0.1	0.00	0.33	0.00	0.42
			0.3	0.00	0.56	0.00	0.44
		2	0.1	0.00	0.43	0.00	0.61
			0.3	0.00	0.55	0.00	0.64
	2	0	0.1	0.00	0.53	0.00	0.45
			0.3	0.00	1.01	0.00	0.47
		2	0.1	0.00	0.56	0.01	0.69
			0.3	0.00	1.05	0.01	0.73
Total Average				**0.00**	**1.48**	0.01	1.48

Table 17 Results of the APTCR and SA_{random} for n= 60 problems

Kruskal-Wallis Test on Relative Difference

Method	N	Median	Ave Rank	Z
APTCR	160	0,000000000	80,5	-15,47
SA_{random}	160	0,896564841	240,5	15,47
Overall	320		160,5	

H = 239,25 DF = 1 P = 0,000
H =273,43 DF = 1 P = 0,000 (adjusted for ties)

Kruskal-Wallis Test on CPU

Method	N	Median	Ave Rank	Z
APTCR	160	0,005000	80,5	-15,47
SA_{random}	160	0,608000	240,5	15,47
Overall	320		160,5	

H = 239,25 DF = 1 P = 0,000
H =240,77 DF = 1 P = 0,000 (adjusted for ties)

Table 18 Comparison of Effectiveness of the Algorithms for n= 60 Problems

The results suggest that APTCR clearly provided better solutions in shorter CPU times in all cases (positive values) when compared to SA_{random} for problems with 12 jobs. Especially for loose d-to-D windows with low job-machine factor (μ =3) and loose due dates, the solution quality performance of SA_{random} is very low similar to the results in the earlier subsection for small size problems. Although SA_{random} found the solutions in less than 1 second, CPU time performance is worse by more than 100 times than APTCR. Table 18 shows that APTCR has outperformed the SA_{random} algorithm in terms of relative difference and CPU times as it is obvious in Table 17.

The average relative difference effectiveness of APTCR algorithm over SA_{random} increases significantly as the number of jobs per machine decreases (from 6 to 3), the due date tightness decreases (α increases from 1.5 to 2), the d-to-D window tightness decreases (γ increases from 0 to 2), and the release time tightness increases (ρ increases from 0.1 to 0.3). The job-machine and due date tightness factors have the highest effects on the average relative difference. The average CPU time effectiveness of both algorithms does not change significantly among the problem factor levels.

4.4.3 Effectiveness of SA_{APTCR}-MetaRaPS Algorithms for Small Size Problems

The performance of the SA_{APTCR} and MetaRaPS algorithms were compared in terms of relative error and computation time over low and high levels of the problem factors. The same 160 unique problem instances that were generated in Section 4.4.1 are solved. Algorithm solutions were replicated 10 times for each instances and the average, minimum, maximum relative errors and CPU times were Table 19 summarizes the results.

The results show that MetaRaPS with 6% average of average relative errors has performed better than SA_{APTCR} 21% average of average relative errors, and it had always less average relative error except for only one instance (μ = 3, α = 2, γ =0, ρ = 0.3). Moreover, MetaRaPS had lower ranges between maximum and minimum relative errors. Although the parameter, number of iteration (I) was set as 2000, the average CPU time of SA_{APTCR} was less than MetaRaPS. Both algorithm have CPU times less than 2.12 sec. for all problem instances and almost equal ranges.

μ	α	γ	ρ	Avg. Relative Error						CPU (sec.)					
				SA_{APTCR}			MetaRaPS			SA_{APTCR}			MetaRaPS		
				Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.
3	1.5	0	0.1	0.05	0.07	0.01	0.02	0.04	0.01	1.46	1.46	1.46	1,54	1,55	1,54
			0.3	0.09	0.17	0.00	0.00	0.00	0.00	1.32	1.32	1.31	1,41	1,42	1,40
		2	0.1	0.06	0.09	0.04	0.02	0.04	0.00	1.39	1.40	1.39	1,49	1,50	1,49
			0.3	0.17	0.23	0.11	0.04	0.07	0.01	1.94	1.95	1.94	2,06	2,06	2,05
	2	0	0.1	0.21	0.44	0.00	0.17	0.22	0.15	1.97	1.97	1.96	2,06	2,07	2,06
			0.3	0.18	0.29	0.00	0.25	0.30	0.20	1.54	1.55	1.54	1,67	1,67	1,67
		2	0.1	0.64	0.76	0.37	0.07	0.18	0.02	1.59	1.60	1.59	1,69	1,69	1,68
			0.3	1.37	1.96	0.92	0.29	0.51	0.09	1.99	2.00	1.98	2,12	2,12	2,11
6	1.5	0	0.1	0.00	0.01	0.00	0.00	0.00	0.00	1.96	1.97	1.96	2,03	2,03	2,03
			0.3	0.03	0.05	0.00	0.00	0.01	0.00	0.99	0.99	0.99	1,06	1,06	1,05
		2	0.1	0.05	0.07	0.03	0.01	0.03	0.00	1.15	1.15	1.14	1,22	1,22	1,22
			0.3	0.08	0.13	0.03	0.03	0.05	0.00	1.45	1.46	1.45	1,56	1,57	1,56
	2	0	0.1	0.01	0.03	0.00	0.02	0.03	0.02	1.51	1.52	1.51	1,59	1,60	1,59
			0.3	0.14	0.25	0.01	0.03	0.07	0.02	1.10	1.10	1.10	1,17	1,17	1,17
		2	0.1	0.07	0.10	0.02	0.02	0.04	0.00	1.25	1.25	1.25	1,34	1,35	1,34
			0.3	0.16	0.22	0.08	0.05	0.09	0.00	1.55	1.56	1.55	1,66	1,66	1,66
Total Average				**0.21**	0.30	0.10	**0.06**	0.11	0.03	**1.51**	1.51	1.51	**1.60**	1.61	1.60

Table 19 Results of the SA_{APTCR} and MetaRaPS for problems with n= 12

Since the relative error data for each combination does not follow a normal distribution according to Anderson-Darling normality test, the statistical significance of performance between algorithms are analyzed via using nonparametric Kruskal–Wallis test in Minitab 15.1 statistical software program. Table 20 shows that there is a statistical difference between the population mean values of SA_{APTCR} and MetaRaPS performance in terms of relative error ($p = 0.007 < \alpha=0.05$). MetaRaPS has better performance because median of relative errors is less than median value of SA_{APTCR} ($0.008 < 0.061$). On the other

hand, SA_{APTCR} has significantly better CPU time performance than MetaRaPS by having a less median value.

The Levene's test (Table 21) shows that there exists significant difference between variances and MetaRaPS algorithm has lower confidence intervals for relative error (p = 0.003 < α= 0.05) indicating that it is more robust for this problem compared to SA_{APTCR}. On the other hand, there is no significant difference between variances for CPU time.

Kruskal-Wallis Test on Relative Error	Kruskal-Wallis Test on CPU
Method N Median Ave Rank Z MetaRaPS 160 0,008370 129,0 -6,10 SA_{APTCR} 160 0,061534 192,0 6,10 Overall 320 160,5 H = 37,17 DF = 1 P = 0,000 H =38,37 DF = 1 P = 0,000 (adjusted for ties)	Method N Median Ave Rank Z MetaRaPS 160 1,617 176,5 3,09 SA_{APTCR} 160 1,505 144,5 -3,09 Overall 320 160,5 H = 9,55 DF = 1 P = 0,002 H = 9,55 DF = 1 P = 0,002 (adjusted for ties)

Table 20 Comparison of Effectiveness of the Algorithms for n= 12 Problems

Test for Equal Variances: Relative Error	Test for Equal Variances: CPU
95% Bonferroni confidence intervals for standard deviations Method N Lower StDev Upper MetaRaPS 160 0,179393 0,202011 0,230867 SA_{APTCR} 160 0,450057 0,506801 0,579195 Levene's Test (Any Continuous Distribution) Test statistic = 9,20; p-value = 0,003	95% Bonferroni confidence intervals for standard deviations Method N Lower StDev Upper MetaRaPS 160 0,307591 0,346373 0,395850 SA_{APTCR} 160 0,298316 0,335928 0,383914 Levene's Test (Any Continuous Distribution) Test statistic = 0,14; p-value = 0,713

Table 21 Test for Equal Variances: Relative Error and CPU versus Method

4.4.4 Effectiveness of SA_{APTCR} -MetaRaPS Algorithms for Large Size Problems

The same parameter sets are used while testing the effectiveness for large size problems with 60 jobs. Since no optimal solutions exist for these problem sizes, the quality of the solution was measured by relative difference from the best of both algorithms to avoid nonnegative performance values.

$$\text{Relative Difference} = \frac{TWT_{Algorithm} - min(TWT_{SA_{APTCR}}, TWT_{MetaRaPS})}{min(TWT_{SA_{APTCR}}, TWT_{MetaRaPS})} \quad (20)$$

The relative difference was calculated for each of the 160 problem instances then the averages were used for each problem instance combination. In (20), $TWT_{SA\text{-}APTCR}$ and $TWT_{MetaRaPS}$ are the average of algorithm solutions for 10 replications. The comparison results for large size problems in terms of average relative difference and average CPU time are given in Table 22.

The results suggest that MetaRaPS clearly provided better solutions in all cases (positive values) when compared to SA_{APTCR} for problems with 60 jobs. CPU times are not so different from times of algorithms for small size problems. Table 23 shows that MetaRaPS has outperformed the SA_{APTCR} algorithm in terms of relative difference in significantly longer average CPU time.

μ	α	γ	ρ	Avg. Relative Difference		Avg. CPU (sec.)	
				SA_{APTCR}	MetaRaPS	SA_{APTCR}	MetaRaPS
3	1.5	0	0.1	0.02	0.03	0.97	1.32
			0.3	0.02	0.02	1.02	1.49
		2	0.1	0.01	0.04	1.73	1.98
			0.3	0.03	0.01	1.69	2.05
	2	0	0.1	0.14	0.03	1.20	1.75
			0.3	0.31	0.01	1.37	1.83
		2	0.1	0.02	0.05	1.76	2.07
			0.3	0.38	0.00	1.66	2.11
6	1.5	0	0.1	0.01	0.05	0.86	0.82
			0.3	0.01	0.05	0.90	1.09
		2	0.1	0.04	0.00	1.23	1.57
			0.3	0.12	0.00	1.27	1.73
	2	0	0.1	0.00	0.09	0.92	1.10
			0.3	0.00	0.07	0.96	1.39
		2	0.1	0.04	0.00	1.38	1.72
			0.3	0.10	0.00	1.43	1.96
Total Average				0.08	0.03	1.27	1.62

Table 22 Results of the SA_{APTCR} and MetaRaPS for problems with n= 60

Kruskal-Wallis Test on Relative Difference					Kruskal-Wallis Test on CPU				
Method	N	Median	Ave Rank	Z	Method	N	Median	Ave Rank	Z
MetaRaPS	160	0,000000000	143,0	-3,38	MetaRaPS	160	1,732	202,5	8,12
SA_{APTCR}	160	0,016623239	178,0	3,38	SA_{APTCR}	160	1,232	118,5	-8,12
Overall	320		160,5		Overall	320		160,5	
H = 11,41 DF = 1 P = 0,001					H = 65,89 DF = 1 P = 0,000				
H = 13,04 DF = 1 P = 0,000 (adjusted for ties)					H = 65,89 DF = 1 P = 0,000 (adjusted for ties)				

Table 23 Comparison of Effectiveness of the Algorithms for n= 60 Problems

CHAPTER 5

AERIAL REFEULING RESCHEDULING PROBLEM METHODOLOGY

There are some difficulties that make ARSP different from an ordinary parallel machine scheduling problem. One of these difficulties is sourced from the dynamic environment of the AR process where disruptions caused by dynamic and unexpected events are common and require updating the existing AR schedule. Although more disruptions may be considered for ARSP, job related disruptive events are studied in this dissertation since they occur much more frequent than other type of disruptions in air operations. Events that have a potential to cause significant disruptions in the AR schedules are interpreted by the arrival of new jobs, departure of an existing job, and changes in job priorities characterized by a combined change of weight and due date. Processing times, due date tightness and d-to-D window tightness are assumed fixed during the scheduling horizon.

Aerial Refueling *Rescheduling* Problem (ARRP) was addressed in this research. Reactive scheduling is the process of modifying an existing schedule to adapt to changes in a production or operational environment (Sun and Xue, 2001). There are generally three rescheduling approaches : continuous, periodic and event-driven (Church and Uzsoy, 1992). Continuous rescheduling takes a rescheduling action each time an event occurs. Periodic rescheduling defines rescheduling points between which any events that occur are ignored until the following rescheduling point. Finally, in the event-driven rescheduling, a rescheduling action is initiated upon an event with potential to cause significant disruption. Both continuous and periodic rescheduling can be viewed as special cases of event-driven rescheduling. In the ARRP, we use continuous rescheduling approach where updating the existing schedule always takes place when an event occurs because all job related events defined in this study are assumed to have a potantial to cause significant disruption.

As was mentioned in the Literature Review Chapter, rescheduling problems mostly consider both a primary measure of schedule performance, as well as some measure of disruption caused by the rescheduling. Similarly, our objective in the ARRP will not only be minimizing total weighted tardiness, but also minimizing schedule instability.

Instability of the AR schedules is defined here as any changes in job starting times on the assigned machine. Then the measure of schedule instability can be defined as the proportion of rescheduled jobs that change machine assignment and/or starting time.

Consideration of these two objectives leads to the formulation of ARRP as a multi objective scheduling problem. The main issues of our multi objective approach are to develop schedules that are satisfactory with respect to both objectives, and to effectively evaluate the trade-off between the objectives.

5.1 Multi Objective Optimization (MOO)

Multi objective optimization (MOO), refers to finding values of decision variables which correspond to and provide the optimum of more than one objective (Rangaiah, 2008). A general MOO problem is stated as follows:

$$\begin{aligned} &\min f(x) = \left[f_1(x)\ f_2(x) \ldots f_k(x) \right]^T \\ &s.t \\ &g_j(x) \le 0; \quad j = 1,2,\ldots,J \\ &h_l(x) = 0; \quad l = 1,2,\ldots,L \end{aligned} \tag{21}$$

where k is the number of objective functions, J is the number of inequality constraints, L is the number of equality constraints, and $x \in E^n$ is a vector of design variables.

A typical MOO problem have many optimal solutions except for problems with non-conflicting objectives in which case only one unique solution is expected (Rangaiah, 2008). Since objective functions generally conflict with each other, a single point that minimizes all objectives simultaneously does not exist. All optimal solutions obtained for MOO problem are equally good in the sense that each one of them is better than the rest in at least one objective. This implies that one objective improves while at least another objective becomes worse when one moves from one optimal solution to another. Consequently, the notion of *Pareto Optimality* is used to describe solutions for MOO problems.

Pareto-Optimality : A point $x^* \in X$ is said to be a *pareto-optimal solution* for the multi-objective problem, if and only if, no other feasible $x \in X$ exists such that $f_i(x) \leq f_i(x^*) \quad \forall i = 1,2,...,k$, and $f_j(x) < f_j(x^*)$ with strict inequality valid for at least one objective *j*.

Alternatively, a point is *weakly Pareto optimal* if it is not possible to move from that point and improve all objective functions simultaneously (Marler and Arora, 2005).

In MOO, *ideal* and *nadir* objective vectors are occasionally used (Rangaiah, 2008). The *ideal* point provides the lower bounds of the Pareto optimal set. On the other hand, components of the *nadir* objective vector are the upper bounds (i.e., most pessimistic values) of objectives in the Pareto-optimal set. One way of determining *ideal* and *nadir* objective vectors is using individually optimum values. If we assume that there are two objectives, the *ideal* bi-objective vector, $[f_1^* \; f_2^*]$ may contain the optimum values of two objectives, when each of them is optimized individually disregarding the other objectives. The *ideal* point, is not normally feasible because of the conflicting nature of the individual objectives. The *nadir* objective vector, $[f_1^N \; f_2^N]$ may contain the values of two objectives when the other objective is optimized individually. For example, *nadir* point for the first objective, which is denoted by f_1^N is the value of $f_1(\mathbf{x})$ when $f_2(\mathbf{x})$ is optimized individually, and f_2^N is the value of $f_2(\mathbf{x})$ when $f_1(\mathbf{x})$ is optimized individually.

5.1.1 MOO Methods

Three basic approaches which are classified according to the stage of imposing the decision maker's preferences, can be used for considering more than one objective and analyzing the results obtained for MOO.

5.1.1.1 A Priori Methods

These methods imply that the decision makers (DM) indicate the relative importance of the objective functions or desired goals before solving the modified problem to obtain a single Pareto optimal point.

Two main approaches exist in the a priori methods: simultaneous approaches and hierarchical approaches. A simultaneous approach contains value function methods which formulate a value function that includes the original objectives and preferences of the DM for optimization and then solving the resulting single objective optimization (SOO) problem. The weighted sum method, which will be discussed in the following subsection is one particular case of value function methods. However, most scalarization methods do not transfer preferences from the DM to the final solution with complete accuracy. Thus, if the solution is not acceptable, the preferences are altered, and the problem is resolved to obtain another Pareto optimal point (Marler and Arora, 2005).

As for the hierarchical approaches, one example would be lexicographic optimization where the DM must arrange the objectives according to their importance for subsequent solution by a SOO method. The secondary, i.e., the less important, criterion is minimized subject to the constraint that the value of the primary, i.e., the more important, criterion is kept at its optimum value (Azizoglu and Alagoz, 2005).

5.1.1.2 A Posteriori Methods

These methods require generating a representation of the entire Pareto optimal set and selecting a single solution from a set of mathematically equivalent solutions that satisfy preferences. This can be achieved by solving a series of MOO problems and consistently varying parameters (such as weights) in order to yield a series of Pareto optimal points (Marler and Arora, 2005). In effect, all *a posteriori* methods provide many Pareto-optimal solutions to the DM, who will subsequently review and select one for implementation. They include population-based methods such as nondominated sorting genetic algorithm and multi-objective differential evolution as well as multi-objective simulated annealing (Rangaiah, 2008).

5.1.1.3 Interactive methods

These methods require interaction with the DM during the solution of the MOO problem. After an iteration of these methods, the DM review the Pareto-optimal solution(s) obtained and determine further changes desired in each of the objectives. These

preferences of the DM are then incorporated into formulating and solving the optimization problem in the next iteration. At the end of the iterations, the interactive methods provide one or several Pareto-optimal solutions. Examples of these methods are interactive surrogate worth trade-off method and the NIMBUS method (Rangaiah, 2008).

Since we use a scalarized objective function for our bi-objective ARRP, the weighted sum method, which is a *posteriori* method that can be considered a special case of value function methods in the *a priori* methods will be explained in detail.

5.1.2 Weighted Sum Method

In the weighted sum method, a convex combination of functions is reformulated and the general MOO problem is stated as follows:

$$\begin{aligned} &\min_{x \in X} \sum_{i=1}^{k} \lambda_i f_i(x) \\ &s.t \\ &g_j(x) \le 0; \quad j = 1,2,...,J \\ &h_l(x) = 0; \quad l = 1,2,...,L \\ &\sum_{i=1}^{k} \lambda_i = 1 \end{aligned} \tag{22}$$

where λ_i is the weighting factor for the i^{th} objective function. Thus, changing the weights' relative values changes the orientation of the contours for the weighted sum. Minimizing the weighted sum can yield all of the Pareto optimal points if $w_i > 0, \forall i = 1,...,k$ with systematic variations under the convexity assumptions (Miettinen 1999). The solution is also unique if the problem is strictly convex (Grodzevich and Romanko, 2006).

The solution of the resulting SOO problem will depend on w that generates an accurate representation of the Pareto optimal set to a greater extent. In some sense, the weights w_i serve as scale factors for the objective functions. Despite the many methods for determining weights, a satisfactory, a priori selection of weights does not necessarily

guarantee that the final solution will be acceptable (Marler and Arora, 2004). In this case, the optimization problem will have to be solved several times, each time with a different w, in order to find several Pareto-optimal solutions.

Normalization in the weighted sum method

Choosing suitable w to find many Pareto optimal solutions is difficult especially when the objective functions for a problem have significantly different orders of magnitude, although the weighting method is conceptually straightforward. Different function normalization methods can help generate an approximation of the Pareto optimal set that is consistent with the weights assigned by the DM. Some possible normalization types are given by (Marler and Arora, 2005).

Normalization by the minimum or the maximum of the objective functions: A common approach to function transformation is given as :

$$f_i^{norm} = \frac{f_i(x)}{\left| f_i^{\min} \right|} \tag{23}$$

where $f_i^{\min}$ represents the minimum value for objective i. This is referred to as the *lower-bound approach*, and it provides a non-dimensional objective function. The lower limit of f_i^{norm} is restricted to positive values when $f_i^{\min} > 0$, whereas the upper value is unbounded. $f_i^{\min}$ can be defined as an *ideal* point such as $f_i^{\min} = f_i(x^*)$where $f_i(x^*)$is the optimum value of objective i, when it is optimized individually disregarding the other objectives. Equation (23) may be modified as follows :

$$f_i^{norm} = \frac{f_i(x) - f_i^{\min}}{\left| f_i^{\min} \right|} \tag{24}$$

Equation (24) is the *alternate lower-bound approach* and it yields non-dimensional objective function values with a lower limit of zero.

Computational difficulties can arise here if the denominator is close to zero. Use of the optimal solutions to individual problems can also lead to very distorted scaling since optimal values by themselves are in no way related to the geometry of the Pareto set (Grodzevich and Romanko, 2006).

As an alternative to the methods discussed earlier, one may use the maximum value of the function in the denominator rather than $f_i^{\min}$. The consequent *upper-bound approach* is shown as follows :

$$f_i^{norm} = \frac{f_i(x)}{f_i^{\max}} \tag{25}$$

where f^{max} represents the maximum value for objective *i*. f^{max} may be determined as the absolute maximum (if it exists) of $f_i(x)$ or as an approximation of the maximum. Alternatively, an approach more conducive to MOO is to define f^{max} *as a nadir* point which was mentioned earlier (Rangaiah, 2008). Equation (25) provides a nondimensional function value such that $f_i^{norm} \leq 1$ with no restriction on the lower value.

Normalization by the differences of maximum and minimum optimal function values: The most robust approach called the *upper-lower-bound approach* to transforming objective functions, regardless of their original range (Yang et al. 1994).

$$f_i^{norm} = \frac{f_i(x) - f_i^{\min}}{f_i^{\max} - f_i^{\min}} \tag{26}$$

The denominator of the formulation interprets the length of the intervals where the optimal objective functions vary within the Pareto optimal set. The normalization schema uses the differences in the optimal function values of *nadir* and *ideal* points mentioned earlier. In this case, f_i^{norm} is bounded by zero and one for both objective functions and is dependent on the accuracy and the method with which f_i^{min} and f^{max} are determined.

Unlike previous approaches, the denominator is guaranteed to be positive. In addition, this is the only approach that constrains the upper and lower limits of f_i^{norm}. Although the former normalizations have proved to be ineffective and are not practical, this normalization provides a relatively robust approach as the objective functions are normalized by the true intervals of their variation over the Pareto optimal set (Grodzevich and Romanko, 2006).

The following scalarized objective function mentioned in (Rangaiah, 2008) is used for our bi-objective ARRP :

$$Min \quad \lambda \frac{f_1(x) - f_1(x^*)}{f_1^{\max} - f_1(x^*)} + (1-\lambda) \frac{f_2(x) - f_2(x^*)}{f_2^{\max} - f_2(x^*)} \tag{27}$$

where, $0 < \lambda < 1$ is the objective weighting factor. We use $[f_1(x^*), f_2(x^*)]$ as the optimal values of individual objectives when they are optimized individually disregarding the other objective and $[f_1^{max}, f_2^{max}]$ as the values of the individual objective when the other objective is optimized individually.

5.2 The Revised MILP Model for ARRP

The MILP model for ARSP should be revised to take into account also the instability objective for ARRP. This revision is made by modifying the objective function and adding new a decision variable, new parameters and constraints in addition to the existing scheduling constraints mentioned in the scheduling section (Section 3.1). The MILP model can be used to find optimal schedules only for jobs which have not been affected by the disruption time provided that input data was adequately set according to the event type and event time. Starting times and machine assignment decision variable values for the initial schedule are used as an input in the revised model.

Indices

i = 0,1,2,3, . . . , n predecessor jobs;

j = 1, 2, 3, . . . , n successor jobs;

$k = 1, 2, 3, \ldots, m$ *machines;*

Parameters

S_j' = starting time of job j in the original schedule.

y_{jk}' = value of assignment variable in the original schedule.

p_j = processing time of job j;

r_j = release (ready) time of job j;

w_j = weight of job j;

d_j = due date of job j;

D_j = deadline of job j;

M = a large positive integer.

F= fixed cost of returning back to base. It must be much larger than $D_j - d_j$ values.

Decision Variables

S_j = starting time of job j in the new schedule;

T_j = piecewise tardiness of job j;

$$x_{ijk} = \begin{cases} 1, & \text{if job } i \text{ precedes job } j \text{ on machine } k; \\ 0, & \text{otherwise.} \end{cases}$$

$$y_{jk} = \begin{cases} 1, & \text{if job } j \text{ is assigned to machine } k; \\ 0, & \text{otherwise.} \end{cases}$$

$$s_j = \begin{cases} 1, & \text{if instability occurs for job } j; \\ 0, & \text{otherwise.} \end{cases}$$

Instability cost occurs when a deviation exists between the starting times in the original and new schedules or when machine assignment decision variable values in the original and new schedules become different.

Constraints

Using starting time decision variable requires modifying the constraints 4a-b, 5 and 9a-c in the MILP model for ARSP to 28a-b, 29, 30a-c for ARRP as follows:

$$S_j + M(1 - \sum_{k=1}^{m} y_{jk}) \geq r_j \quad j = 1,...,n \tag{28a}$$

$$S_j + M(1 - x_{ijk}) \geq S_i + p_i \quad i = 1,....,n \quad j = 1,...,n \quad k = 1,...,m \tag{28b}$$

$$S_j + p_j \leq D_j \cdot \sum_{k=1}^{m} y_{jk} \quad j = 1,...,n \tag{29}$$

$$T_j \geq S_j + p_j - d_j \quad j = 1,...,n \tag{30a}$$

$$T_j \geq F(1 - \sum_{k=1}^{m} y_{jk}) \quad j = 1,...,n \tag{30b}$$

$$S_j, T_j \geq 0 \quad j = 1,...,n \tag{30c}$$

Secondly, some constraints are needed to handle the instability objective. Suppose that a deviation vector, δ is formed by the deviation between the original and new starting times (S_j - S_j^1) and the deviation between the assignment decision variables (y_{jk} -y_{jk}^1). All possible cases for each job j and machine k are given below. Although some of these cases can be redundant, they are still listed to explain how the constraints would work in all cases, and not only for cases that involve instability:

Case 1: Job j which was assigned to machine k in the original schedule (y_{jk}^1=1) is not assigned to another machine (y_{jk} = 0) with the same starting times. Then δ = ((S_j = S_j^1), (y_{jk} -y_{jk}^1)) = (0, -1).

Case 2: Job j which was assigned to machine k in the original schedule($y_{jk}'=1$) is then assigned to another machine ($y_{jk} = 0$) but with earlier starting time, ($S_j < S_j'$). Then δ = (-, -1) meaning that subtraction of the starting times takes a negative value.

Case 3: Job j which was assigned to machine k in the original schedule ($y_{jk}'=1$) is then assigned to another machine ($y_{jk} = 0$) but with later starting time, ($S_j > S_j'$). Then δ = (+, -1).

Case 4: Job j which was assigned to machine k in the original schedule ($y_{jk}'=1$) is still assigned to machine k ($y_{jk} = 1$) but with an earlier starting time, ($S_j < S_j'$). Then δ = (-, 0).

Case 5: Job j which was assigned to machine k in the original schedule ($y_{jk}'=1$) is still assigned to machine k ($y_{jk} = 1$) but with a later starting time, ($S_j > S_j'$). Then δ = (+, 0).

Case 6: Job j which was assigned to machine k in the original schedule ($y_{jk}'=1$) is still assigned to machine k ($y_{jk} = 1$) with the same starting time, ($S_j = S_j'$). Then δ = (0, 0).

Case 7: Job j which was not assigned to machine k in the original schedule ($y_{jk}'=0$) is then assigned to machine k ($y_{jk} = 1$) with the same starting time, ($S_j = S_j'$). Then δ = (0, 1).

Case 8: Job j was not assigned to machine k in the original schedule ($y_{jk}'=0$) is then assigned to machine k ($y_{jk} = 1$) with an earlier starting time, ($S_j < S_j'$). Then δ = (-, 1).

Case 9: Job j was not assigned to machine k in the original schedule ($y_{jk}'=0$) is then assigned to machine k ($y_{jk} = 1$) with a later starting time, ($S_j > S_j'$). Then δ = (+, 1).

Case 10: Job j was not assigned to machine k in the original schedule ($y_{jk}'=0$) and is not assigned to machine k ($y_{jk} = 0$) and the starting times are the same, ($S_j = S_j'$). Then δ = (0, 0).

Case 11: Job j was not assigned to machine k in the original schedule ($y_{jk}'=0$) and is not assigned to machine k ($y_{jk} = 0$) but the starting time is earlier, ($S_j < S_j'$). Then δ = (-, 0).

Case 12: Job j was not assigned to machine k in the original schedule ($y_{jk}'=0$) and is not assigned to machine k ($y_{jk} = 0$) but the starting time is later ($S_j > S_j'$). Then δ = (+, 0).

It is clear that ($S_j - S_j'$) takes *(+, 0, -)* values and ($y_{jk} - y_{jk}'$) takes *(+1, 0, -1)* values. According to our definition, $s_j = 0$ only when there is no change to the starting times ($S_j - S_j' = 0$) *and* no change to machine assignment ($y_{jk} - y_{jk}' = 0$). That is:

$$s_j = \begin{cases} 1, & \text{if } S_j - S'_j \neq 0 \text{ or } y_{jk} - y'_{jk} \neq 0 \\ 0, & \text{otherwise, } S_j - S'_j = 0 \text{ and } y_{jk} - y'_{jk} = 0 \end{cases}$$

Consequently, new constraints (31) handle the above twelve cases, can be added to the model.

$$M.s_j \geq \left|(S_j - S'_j)\right| + \left|(y_{jk} - y'_{jk})\right| \qquad j = 1,...,n \qquad k = 1,...,m \tag{31}$$

where M is a large integer number to define the binary instability decision variable. However these constraints cause a non-linearity in the model, since they contain absolute valucs. Thus, two groups of alternative linear constraints (32a,b) can be used instead of the above constraint.

$$M.s_j \geq \theta(S_j - S'_j) + (y_{jk} - y'_{jk}) \qquad j = 1,...,n \qquad k = 1,...,m$$

$$M.s_j \geq -\theta(S_j - S'_j) - (y_{jk} - y'_{jk}) \qquad j = 1,...,n \qquad k = 1,...,m \tag{32a,b}$$

where θ is a positive integer coefficient that is large enough (e.g. θ = 10) to avoid a zero right hand side values when the deviation values of the starting times and assignment decision variable have opposite signs (e.g. when $(S_j - S_j) = 1$ and $(y_{jk} - y_{jk}') = -1$, constraints (32a-b) assign $s_j = 1$).

The objective is to minimize a linear convex combination of normalized *TWT* and *Instability* objectives, where each objective has an allocated weight indicating its importance.

The objective function of ARRP is as follows:

$$\min Z = \lambda \frac{\sum_{j=1}^{n} w_j T_j - f_{TWT}(x^*)}{f_{TWT}^{\max} - f_{TWT}(x^*)} + (1-\lambda) \frac{\sum_{j=1}^{m} s_j - f_{Instability}(x^*)}{f_{Instability}^{\max} - f_{Instability}(x^*)} \quad (33)$$

where $0 < \lambda < 1$. Jobs that have been completed before the event time have to be excluded from the problem while implementing the MILP model for ARRP. On the other hand, the jobs which are in process when a disruptive event occurs are included without any tardiness or instability cost. They are automatically assigned to the first places of the corresponding machines such that their completion times indicate the earliest available times for each machine. Since available times are not zero anymore, unscheduled jobs can start processing after the completion of processing the jobs in the original schedule. In order to ensure that they do not have any impact on the objective function value, data for the initial scheduling problem should be adjusted for the rescheduling problem as follows. For every job j that is in-process when a disruption occurs, we set $r_j = 0$; $w_j =$ *a large integer;* $p_j = d_j = D_j =$ completion time of job j. Consequently, the model will be forced to assign these jobs to the same machine by starting at the same time as the original schedule. Note that maximum index values for TWT and Total Instability, n and m need not to be equal. For example, if the event is the arrival of a new job, this job is not included in calculating the instability. Otherwise, there will be bias not to schedule the new job whose initial machine assignment decision variable value is zero. Only weighted tardiness of the new job has an effect on the composite objective function.

The ideal and nadir objective vectors ($[f_{TWT}^{*}\ f_{Instability}^{*}]$ and $[f_{TWT}^{max}\ f_{Instability}^{max}]$) can be used to normalize the objective values. In order to find these values, it is required to implement the MILP model twice by using adequate λ values ($0 < \lambda < 1$) and theoritical pessimistic and optimistic objective values for an individual objective that give the possible largest objective value ranges. For example, f_{TWT}^{*} and $f_{Instability}^{max}$ can be found after the first run by setting as follows:

$\lambda = 1 - \varepsilon$, where ε is a very small positive real number,

f_{TWT}^{min} = 0, optimistic *TWT* value of the case that all jobs are completed before their due dates.

f_{TWT}^{max} = pessimistic *TWT* value of the case that all jobs missed their deadlines.

$f_{Instability}^{min}$ = 0, *optimistic Total Instability* value of the case that all jobs start on the same machine at the same time according to the original schedule.

$f_{Instability}^{max}$ = *pessimistic Total Instability* value of the case that all jobs' starting time and machine assignment are changed.

Similarly, $f_{Instability}^{*}$ and f_{TWT}^{max} can be found after the second run by setting $\lambda = \varepsilon$

In order to clarify the methodology of obtaining the rescheduling optimal solution, a sample data for the problem with 12 jobs and 3 machines is given as follows:

Jobs = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
Release Times = [4.7, 4.2, 18.7, 6.0, 18.8, 3.8, 10.5, 11.3, 3.5, 16.3, 19.0, 5.3],
Processing Times = [40, 15, 16, 24, 15, 16, 22, 39, 20, 41, 22, 36],
Weights = [9, 7, 1, 9, 2, 5, 1, 2, 5, 4, 5, 8],
Due Dates = [84.7, 34.7, 50.7, 54.00, 48.8, 35.8, 54.5, 89.3, 43.50, 98.3, 63.0, 77.3],
Deadlines = [124.7, 49.7, 66.7, 78.00, 63.8, 51.8, 76.5, 128.3, 63.5, 139.3, 85.0, 113.3],
Fixed Unavailability Cost = 120

For the above scheduling problem instance, the best total weighted tardiness value as obtained by solving the MILP is 161.50 for the original schedule shown in Figure 8 with the following starting times and machine assignment decision variable values for the 12 jobs:

Starting times = [47.5, 4.2, 34.2, 23.5, 19.2, 3.8, 50.2, 77.8, 3.5, 72.2, 19.8, 41.8],

Machine Assignment = [[1 0 0], [0 0 1], [0 0 1], [1 0 0], [0 0 1], [0 1 0], [0 0 1], [0 1 0], [1 0 0], [0 0 1], [0 1 0], [0 1 0]].

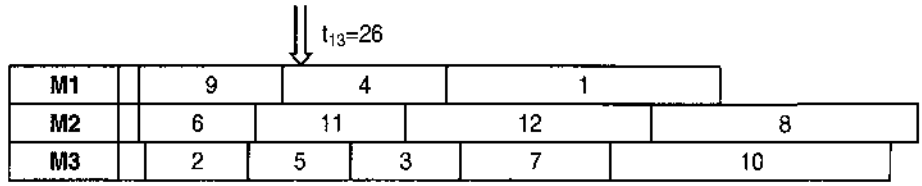


Figure 8 Sample Original Solution

Suppose that a new job with $p_{13} = 30$, $w_{13} = 5$, $d_{13} = 86$, and $D_{13} = 116$ arrives at time, $r_{13} = 26$. The starting time and machine assignment decision variable values are set 0 for the new job. Since jobs 2, 6 and 9 had already completed, they are removed from the job set that needs rescheduling. Jobs 4, 5 and 11 were in-process at machines 1, 3 and 2 respectively at the arrival time of job 13. These jobs are used to determine the first available times of each machine to reschedule. So, for these processing jobs, we set $r_j = 0$; w_j= a large integer (eg. 999); $p_j = d_j = D_j$ = completion time of the job j in the original schedule. As a result, the new data set including the newly arrived job (first element of the vector) is given as follows :

Jobs = {13, 1, 3, **4, 5,** 7, 8, 10, **11**, 12}
Release Times = [26, 4.7, 18.7, **0, 0**, 10.5, 11.3, 16.3, **0**, 5.3],
Processing Times = [30, 40, 16, **47.5, 34.2**, 22, 39, 41, **41.8**, 36],
Weights = [5, 9, **1**, **999**, **999**, 1, 2, 4, **999**, 8],
Due Dates = [86, 84.7, 50.7, **47.5, 34.2**, 54.5, 89.3, 98.3, **41.8**, 77.3],
Deadlines = [116, 124.7, 66.7, **47.5**, **34.2**, 76.5, 128.3, 139.3, **41.8**, 113.3],
Old Starting Times = [0, 47.5, 34.2, 23.5, 19.2, 50.2, 77.8, 72.2, 19.8, 41.8],
Old Machine Assignment = [[0 0 0], [1 0 0], [0 0 1], [1 0 0], [0 0 1], [0 0 1], [0 1 0], [0 0 1], [0 1 0], [0 1 0]].

For this case, in order to find the vectors ($[f_{TWT}^{*}\ f_{Instability}^{*}]$ and $[f_{TWT}^{max}\ f_{Instability}^{max}]$), we firstly set the initial values as follows:

$\lambda = 0.9999$ where $\varepsilon = 0.0001$ regarding the total weighted tardiness objective.

$f_{TWT}^{min} = 0$, where there is no tardiness cost,

$f_{TWT}^{max} = 10\text{x}5\text{x}120 = 6000$, where all 10 jobs with average weight miss their deadlines,

$f_{Instability}^{min} = 0$, where there is no instability,

$f_{Instability}^{max} = 9$ (the new job is excluded), where starting times or machine assignments are changed for all old jobs.

In order to simplify the pessimistic TWT calculation, an estimate is used by assuming that all jobs with average weight miss their deadlines. The reason behind using an average weight is that since some weight values have been already reset after the job arrival, it is confusing to use actual weights. Moreover, the pessimism level does not change the best solution of the TWT objective where Total Instability objective has almost zero effect on the objective value. We found TWT = 284.3 and Instability = 2 for the best value of the combined objective with above optimistic and pessimistic values. Secondly, we set $\lambda = 0.0001$ keeping above values and found TWT = 761.5 and Instability = 0. Therefore, $[f_{TWT}^{*}\ f_{Instability}^{*}] = [284.3\ \ 0]$ and $[f_{TWT}^{max}\ f_{Instability}^{max}] = [761.5\ 2]$. Consequently, the objective function for this problem is as follows:

$$\min Z = \lambda \frac{\sum_{j=1}^{n} w_j T_j - 284.3}{761.5 - 284.3} + (1-\lambda)\frac{\sum_{j=1}^{m} s_j - 0}{2-0}$$

$$= \lambda \frac{\sum_{j=1}^{n} w_j T_j - 284.3}{477.2} + (1-\lambda)\sum_{j=1}^{m} s_j / 2 \qquad (34)$$

The optimal solution obtained for the above ARRP by using the objective function (34) with $\lambda = 0.5$ is shown in Figure 9. This solution has the optimal objective value of 0.32 with TWT = 466.92 and Instability = 1.

M1		9		4		13		1	
M2		6		11		12		8	
M3		2	5		3		7		10

Figure 8 Optimal Rescheduling Solution by using $\lambda = 0.5$

In the Computational Study chapter, experiments will be conducted with the various values of objective weighting factor (λ) for extended analysis.

5.2 Rescheduling Algorithms for ARRP

Generally three rescheduling methods exist to repair the disrupted schedule: right shift rescheduling, partial rescheduling and regeneration. Right shift rescheduling postpones each remaining operation by the amount of time needed to make the schedule feasible. Partial rescheduling algorithm reschedules only the operations affected directly or indirectly by the disruption. Regeneration reschedules the entire set of operations not processed before the rescheduling point, including those not affected by the disruption (Church and Uzsoy, 1992).

In this dissertation, we developed reactive scheduling mechanisms to update instantly the current schedule at the disruption time. Recall that following three job related events are the main reasons of schedule disruptions in this research: arrival of new jobs, departure of an existing job, and changes in priority (regarding weight, due date and deadline parameters) for an existing job. All three repair methods mentioned above are employed to develop our five repair algorithms:

- Complete regeneration algorithm, MetaRE (MetaRaPS Regeneration) by using MetaRaPS with APTCR rule and the objective function that was defined earlier as the combination of TWT and Instability objective,
- Complete regeneration algorithm, SEPRE (Sequence Preserving Regeneration) by using APTCR rule and preserving the initial sequence of jobs on each machine,
- Best (TWT+Instability) insertion algorithm, BestINSERT with right shift,
- Left shift algorithm, LSHIFT,
- Partial rescheduling algorithm (SHUFFLE) by shuffling jobs according to APTCR rule.

All these heuristic algorithms consider the tardiness objective by taking into account piecewise tardiness cost with due date to deadline (d-to-D) windows, job release times and also the schedule stability objective. Algorithm usage matrix is shown in Table 24.

	MetaRE	SEPRE	BestINSERT	LSHIFT	SHUFFLE
Arrival of a new job	√	√	√		
Departure of an existing job	√			√	
Changes in priority	√		√		√

Table 24 Disruption-Rescheduling Algorithm Usage Matrix

All algorithms require an initialization phase to determine the rescheduling point and the set of rescheduling jobs that are affected by the disruption. The procedure below can be used to obtain this initial data for the proposed algorithms. The second repair phase may differ according to the disruption type.

Determine-JobSet-to-Reschedule

S: set of jobs in the initial schedule

U: set of jobs to be rescheduled

Data : release time (r_j), processing time (p_j), weight (w_j), due date (d_j), deadline (D_j) input.

θ^{old} *:* Initial solution

t_{event} : disruptive event occurrence time

t : decision time for the following assignment

M = set of machines = {1,2,…,m}

$Machine_j$: index of the machine to which job j was assigned,

$CompTime_j$: completion time of job j

$Cmax_i$: makespan of machine i

1. Initialize *Data* for $\forall j \in S$ and for the new job (in the case of new job arrival).
2. Set $CompTime_j^{old}$, $Machine_j^{old}$ and $\theta^{old} = \{ \theta_1^{old}, \theta_2^{old}, \ldots, \theta_j^{old} \}$
   where $\theta_j^{old} = (CompTime_j^{old}, Machine_j^{old}) \forall j \in S$
   (Set $CompTime_{new}$ = large integer, $Machine_{new}$ =m+1 in case of new job arrival)
3. Determine $Cmax_i$ = first completion time after t_{event} $\forall i \in M$
4. Set $t = \min_{i \in M} \{ Cmax_i \}$

5. Set $U = \{U \subseteq (S \cup \text{new job}^*) : CompTime_j - p_j \geq t \quad \forall j \in U\}$

The event occurance time, t_{event} is defined as the decision time of the job's departure or priority change events. For example, job j is decided to cancel at time $t_{decision}$ even it was scheduled to start processing at time t_{start}, $t_{decision} < t_{start}$. In this case, t_{event} is defined as $t_{decision}$ instead of t_{start}. Similarly, priority changes are assumed to be decided before the start time of the corresponding job. This assumption enlarges the number of affected jobs.

An example is given in Figure 10 to explain this assumption for the instance with number of jobs = 12, number of machines = 3. Assume that a decision to change the priority parameters for randomly selected job 10 was made at time $t_{decision}$ = 35. The set of affected jobs (unshaded jobs) is given in Figure 11. Note that earliest available times for each machine are different from zero and each other ($Cmax_1$=50, $Cmax_2$=40, $Cmax_3$=60) after implementing the procedure as seen in Figure 11.

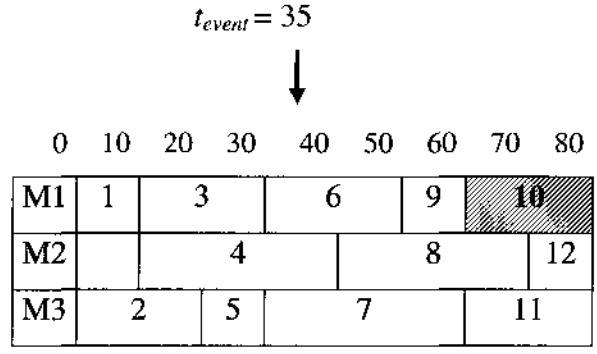


Figure 10 An Initial Schedule to Change Job Priority

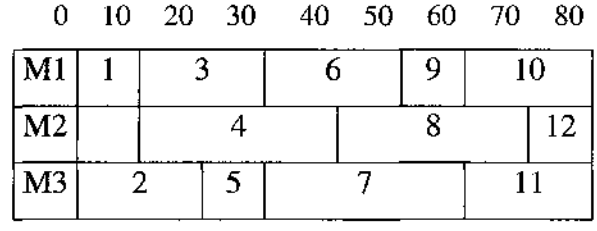


Figure 11 Affected Jobs to be Rescheduled

5.2.1 MetaRE Algorithm

MetaRE is a complete regeneration repair algorithm that reschedules all jobs which have not started by disruption time. It has two phases: initialization phase to determine the rescheduling set and a rescheduling phase using MetaRaPS algorithm. The main algorithm of the second phase is not different from the MetaRaPS metaheuristic which was successfully implemented earlier by using the proposed APTCR rule for ARSP. Moreover, MetaRE considers multi objectives (TWT and Instability) to compare the best solution with neighbor solutions generated by the same combined objective function defined as (33). MetaRE algorithm is given below where t_{max} is the number of iterations until termination and θ^{best} is the best solution found until the last iteration.

MetaRE_Algorithm

1. Execute Determine-JobSet-to-Reschedule
2. Set $CompTime_j = 0$, $Machine_j = m+1$, $\theta^{best} = (CompTime_j^{best}, Machine_j^{best}) \; \forall \; j \in U$
   a. $f_{best} = 1$, $f_{worst} = 0$, $Memory = (f_{best}, f_{worst}, \theta^{best})$, $iteration = 0$.
3. **While** $iteration < t_{max}$
4. Execute MetaRaPS_Algorithm by using (8), (10) and θ^{old}
5. Update *Memory*
6. $iteration = iteration + 1$
7. **end While**
8. Display the f_{best} and θ^{best}

This algorithm reschedules the affected jobs by the MetaRaPS algorithm (recall Section 3.3.3) with APTCR rule (recall Section 3.3.1). After determining the set of affected jobs (Step 1), it constructs a feasible solution by repeatedly adding randomly selected jobs from a candidate list (Steps 3-7). This list is formed according to the APTCR rule priority index values and MetaRaPS parameters (the priority percentage (p%) and the restriction percentage (r%)). Finally, an improvement phase is applied in MetaRE by using a local search algorithm only to the constructed solutions with promising values of the multi

objective function (TWT and Instability objective) compared to the best and worst objective values (See Section 3.3.3 for a detailed example on MetaRaPS). Initial values for the best and worst objective can be defined as 1 and 0 respectively, which are the extreme values of the combined objective function. The best and worst objective values and the best schedule found until the last iteration are updated after each iteration, until the stopping criterion is satisfied.

5.2.2 SEPRE Algorithm

SEPRE is also a complete regeneration repair algorithm based on APTCR rule with two phases like MetaRE. After determining rescheduling set, jobs are assigned repeatedly using APTCR priority index. Additionally, SEPRE takes into account the original sequences on each machine as in the Affected Operation Rescheduling (Abumaizar and Svetska, 1997). Regarding stability, the initial schedule which is assumed to hold the best TWT performance among other feasible solutions, should be preserved. The basic principle behind the concept of Affected Operation Rescheduling is to accommodate any disruption by pushing some operation starting times forward (delaying them) by the minimum amount required to: (1) keep the technological constraints satisfied and (2) preserve the initial sequence of operations on each machine. This guarantees that the robustness of the initial schedule is preserved as much as possible by minimizing the delay (starting time deviation) and reducing the sequence deviation to zero. SEPRE algorithm is given below.

SEPRE_Algorithm

1. Execute Determine-JobSet-to-Reschedule
2. Set $CompTime_j = 0$, $Machine_j = m+1$, $\forall\, j \in U$
3. **while** $U \neq \emptyset$
4. Calculate the index $\pi j(t)\ \forall\, j \in U$ by using APTCR
5. Sort jobs by $\pi j(t)\quad \forall\, j \in \{j \in U \mid \pi j(t) > 0\}$
6. Set r = 1
7. Find r^{th} ranked job and $Machine_j^{old}\quad j \in sorted\ U$

8. **if** $CompTime_j^{old} < CompTime_k^{old}$ $\forall$ k $\in \{k \neq j \in U \mid Machine_k^{old} = Machine_j^{old}\}$ **then**
9. Find $i = \{i \in M : Cmax_i(t) = \min_{m \in M}\{Cmax_m\}\}$
10. Update $CompTime_j = Cmax_i + max(r_j, t) + p_j$ and remove j from U
11. Update $Cmax_i = CompTime_j$
12. Update $t = \min_{i \in M}\{Cmax_i\}$
13. **end if**
14. **else** r = r +1 and Go to Step 7
15. **end while**
16. Calculate the combined objective value for U.

This algorithm reschedules the affected jobs one by one using APTCR priority index by taking into account the original sequences on each machine. For example, assume that the initial schedule and the set of affected jobs are given like in Figure 11. Additionally, a new job 13($p_{13} = 20$) arrives at time t=35.

Step 3. The set of unscheduled jobs, U ={8, 9, 10, 11, 12, 13}

Step 4-5. Assume that priority index values are calculated as in Section 3.3.1 and are ranked for t= 40 as follows:

$\pi_{10}(t) = 0.609 > \pi_9(t) = 0.262 > \pi_{13}(t) = 0.166 > \pi_8(t) = 0.152 > \pi_{11}(t) = 0.151 > \pi_{12}(t) = 0.051$

Step 6-7. r = 1; meaning find the highest ranked job, which is job 10.

Step 8. $CompTime_{10}^{old}$ is not less than $CompTime_9^{old}$ where k= 9. So set r = 2 and go to Step 7.

Step 7. r = 2 meaning find the second highest ranked job which is job 9.

Step 8. $CompTime_9^{old} < CompTime_{10}^{old}$ where k= 10.

Step 9. $Cmax_1 = 50$, $Cmax_2 = 40$, $Cmax_3 = 60$; i = 2 (randomly selected)

Step 10-12. Assign job 9 to machine 2. $CompTime_9 = 40 + 10 = 50$.

$U = \{10,11,12,13\}$, $Cmax_2 = 50$, $t = min\{50, 50, 60\} = 50$. Go to Step 3.

The final schedule found by SEPRE is given in Figure 12.

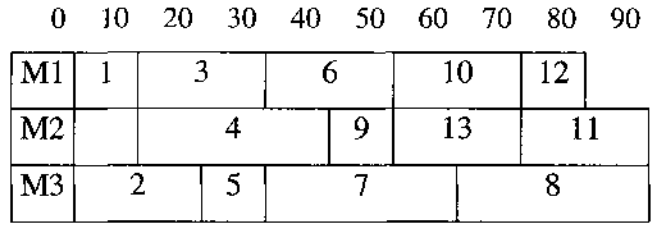


Figure 12 Rescheduled jobs by SEPRE

5.2.3 BestINSERT Algorithm

BestINSERT is a partial and right shift repair algorithm which assumes that only one machine is affected by a disruption. BestINSERT tries all job insertion alternatives for the arriving job or the changed job priority to find the best insertion with minimum value of combined objective function discussed earlier (TWT and stability). Thus, BestINSERT can emphasize both efficiency and stability objectives. After inserting the job into a place on one of the machine, all remaining jobs assigned to that machine are shifted by the amount of processing time of inserted job. BestINSERT keeps the sequence on the corresponding machine and keeps the completion times and job assignments on the remaining machines. BestINSERT algorithm is given below.

BestINSERT _Algorithm

I : Insertion set after the earliest available time.

f_i : Combined objective function value after the insertion into place i and performing a right shift.

f^{best} : Best TWT value found so far.

θ^{best} : Schedule with f^{best}

1. Execute Determine-JobSet-to-Reschedule
2. Set $f^{best} = 1$ and $\theta^{best} = \theta^{old} \quad \forall j \in U, i = I$
3. **while** $I \neq \emptyset$
4. Insert the new job into place $i \in I$
5. Update $CompTime_j = CompTime_j^{old} + p_{new} \; \forall j$ on the right

6. Calculate f_i
7. **If** $f_i < f^{best}$ **then** Update f^{best} and θ^{best}
8. Remove i from I and $i = i+1$
9. **End While**
10. Display the combined objective value for θ^{best}.

The earlier example can be used again to illustrate the BestINSERT algorithm. For a new job 13 arriving at time t=35, there exist 8 insertion alternatives between jobs 6 and 9, 9 and 10, 4 and 8, 8 and 12, 7 and 11 and after jobs 10, 11, 12.

Step 3. I = {6-9, 9-10, 4-8, 8-12, 7-11, 10-, 11-, 12-}
Step 4-9. Compare the combined objective function value for i = 1, 2, ..., 8 and select the best one. Some insertion examples are given in Figure 13, Figure 14, and Figure 15.

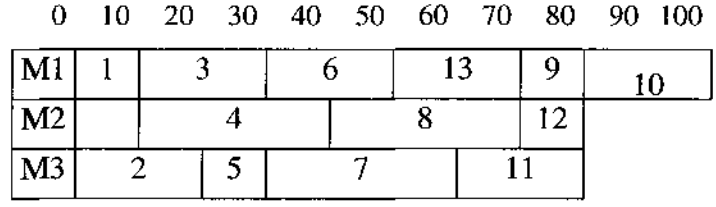


Figure 13 Insert job 13 between jobs 6 and 9 (i = 1)

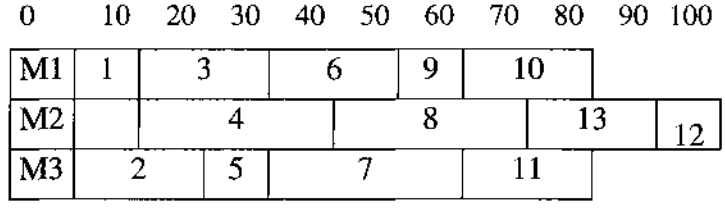


Figure 14 Insert job 13 between jobs 8 and 12 (i = 4)

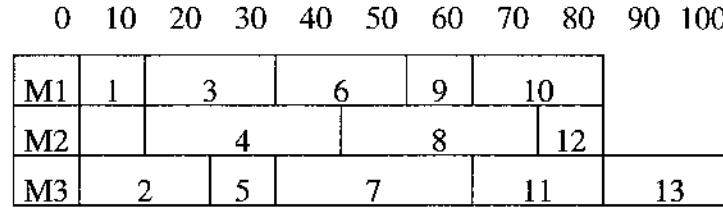


Figure 15 Insert job 13 after job 11(i = 7)

BestINSERT can also be used for priority change events. In this case, firstly the job with priority change can be taken out of the old schedule and left shifting the jobs on its right and secondly all insertion alternatives are attempted for that job like the case of the arrival of a new job.

5.2.4 LSHIFT Algorithm

LSHIFT is also a partial and left shift repair algorithm that is applied only to a particular machine in a similar way like BestINSERT. After taking out a job from its place on a machine, all remaining jobs assigned on that machine are preponed (scheduled earlier) by the amount of processing time (p_{depart}) of the departing job. LSHIFT keeps completion times and job assignments on the other machines as they are, and keeps the job sequence on the machine with the departing job as is. LSHIFT algorithm is given below.

LSHIFT_Algorithm

1. Execute Determine-JobSet-to-Reschedule
2. Select a random job $j \in U$ and $Machine_j^{old}$
3. Remove j from U
4. Update $CompTime_j = CompTime_j^{old} - p_{depart} \; \forall \; j$ *on the right at* $Machine_j^{old}$
5. Find Cmax, the latest completion time at $Machine_j^{old}$
6. While Cmax $\leq D_j - p_j, j \in \{k \in S | \; Machine_k^{old} = m+1\}$, Do
   $CompTime_j$= Cmax + p_j
7. Calculate the combined objective value for U.

Note that in Step 6, LSHIFT checks the possibility of scheduling jobs that were in the unscheduled list of jobs in the original schedule. One of the unscheduled jobs (job j) in the old schedule can possibly be scheduled if $(D_j - p_j)$ is grater than the completion time of the last job on the rescheduled machine after left shifting. Note also that the departing job is excluded from the calculation of the instability objective. To give an example of how LSHIFT works, assume that the initial schedule and the set of affected jobs are given in Figure 16 as given in Figure 10. But this time, the departure of job 8 which was randomly selected among the existing jobs, is decided at time $t_{decision} = 35$. Moreover, there exists an unscheduled job 16 (with processing time $p_{14} = 20$ and deadline $D_{14} = 70$) according to the original schedule decision.

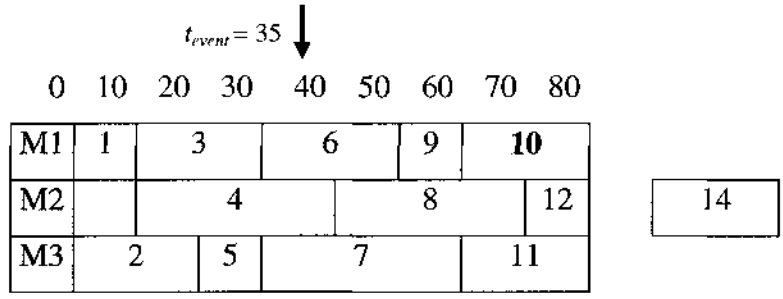


Figure 16 An Initial Schedule for job departure

After the first three steps, the set of jobs to reschedule is determined as U = {9, 10, 11, 12, 14}. Consequently, the final schedule after left shifting and accepting the jobs (because $CompTime_{12} = 80 - 30 = 50$ and $CompTime_{14} = CompTime_{12} + p_{14} \leq D_{14} = 70$) after left shifting) is given in Figure 17.

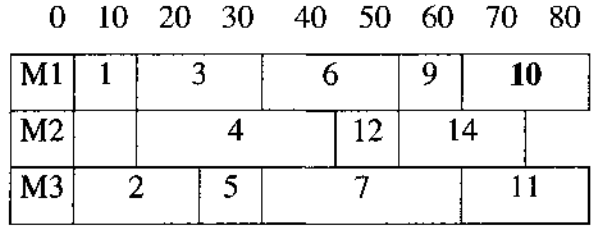


Figure 17 Rescheduled jobs by LSHIFT

5.2.5 SHUFFLE Algorithm

SHUFFLE is a partial repair algorithm that is applied to one of the machines assuming that a changing priority disruption affects only that particular machine. It is based on APTCR priority index where the pairwise swapping decisions on a particular machine are made repeatedly to achieve the APTCR priority index ranking calculated by using the new priority parameters(due date, deadline and weight) of the disrupted job. SHUFFLE algorithm is given below.

SHUFFLE_Algorithm

R : set of successor jobs

L : set of predecessor of job

1. Execute Determine-JobSet-to-Reschedule
2. Select a random job $j \in U$ and $Machine_j^{old}$,
3. Update w_j, d_j and D_j, $t = StartTime_j$
4. **While** $R \neq \emptyset$
5. Calculate $\pi_j(t)$ *and* $\pi_k(t)$ job by using APTCR, $k \in R$ of job j on $Machine_j^{old}$
6. **If** $\pi j(t) < \pi_k(t)$ **then** Swap and update t and R
7. **Else** Terminate
8. **end While**
9. **While** $L \neq \emptyset$
10. Calculate $\pi_j(t)$ *and* $\pi_k(t)$ job by using APTCR, $i \in L$ of job j on $Machine_j^{old}$
11. **If** $\pi j(t) > \pi_i(t)$ **then** Swap and update t and R
12. **Else** Terminate
13. **end While**
14. Check possibility of scheduling the unscheduled jobs in the old schedule
15. Calculate the combined objective value for U.

In order to consider the stability objective in addition to TWT objective, SHUFFLE changes the original schedule in small steps by swapping two neighbor jobs according to their priority index values. However the small steps to the right are firstly attempted to reduce the number of affected jobs. If it is not possible to shuffle to the right then swapping trials to the left are applied. The priority disruption is assumed to occur only at jobs which are scheduled in the old schedule so that SHUFFLE can be implemented. Otherwise, only insertion and complete regeneration would be enough to repair the schedule.

In order to describe the SHUFFLE procedure, the previous example data for LSHIFT can be used for priority change disruption (Figure 18). Assume that due date and deadline of job 12 are decided at time $t_{decision} = 35$ to be made tighter (with fixed d-to-D window) and set its weight as the highest value of 9 over 9. Moreover, there exists an unscheduled job 14 (with processing time $p_{14} = 20$ and deadline $D_{14} = 70$) according to the original schedule decision.

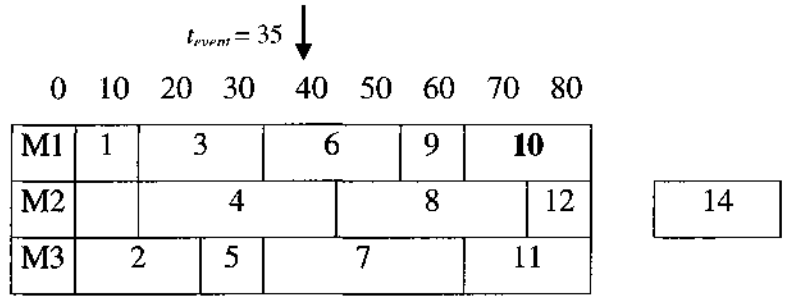


Figure 18 An initial schedule for job priority disruption

The set of jobs to reschedule is determined as U = {8, 9, 10, 11, 12, 14}. First, since $R = \emptyset$, go to Step 9. $L = \{8\}$ and for t = 40, priority index values are compared for jobs 8 and 12. Since $\pi_8(t) < \pi_{12}(t)$, jobs 8 and 12 are swapped. Consequently, $t = StartTime_{12} = 40$, R = {12} and $L = \emptyset$. Assume that deadlines of both jobs 8 and 14 are $D_8 = D_{14} = 70$. Thus, job 8 cannot be scheduled anymore. Finally, job 14 can be scheduled instead of job

8 because $CompTime_{14} = CompTime_{12} + p_{14} = 50 + 20 \leq D_{14}$. Eventually, the updated schedule is given in Figure 19.

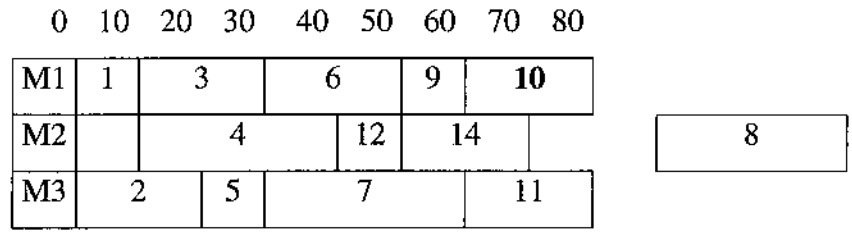


Figure 19 Updated schedule by SHUFFLE

CHAPTER 6

COMPUTATIONAL STUDY FOR ARRP'S RESCHEDULING ALGORITHMS

In order to measure the effectiveness of the rescheduling algorithms, they are compared to optimal solutions for small size problems (n=12) obtained using the MILP model discussed earlier. For large size problems (n=60), the algorithms are compared to each other. The same look-ahead parameter estimates and MetaRaPS parameter values that were set in the Section 4.3 are used for problem instances.

6.1 Data Generation

1. For small size problems, the number of jobs is set to value n = 12 and the number of machines m= 3. For large size problems, the number of jobs is set to value n = 60 and the number of machines m= 10.

2. The processing times p_j are uniformly distributed integers in the interval [15, 45]. The processing time of the new job p_{new} is set to the middle value of 30.

3. The weights for the jobs w_j are uniformly distributed integers in the interval [1, 9]. The weight of the new job w_{new} is set to the middle value of 5.

4. Given the average of jobs' processing times ($\bar{p}$), the average of the jobs' release times ($\bar{r}$), job machine factor (μ), and coefficient (ϕ), which takes into account the effect of the release times on the makespan (assumed here $\phi = 0.1$), then $\hat{C}_{max} = (\phi\bar{r} + \bar{p}).\mu$ estimates C_{max} of the problem. A similar approach was used by Lee and Pinedo (1997).

5. Release times r_j are uniformly distributed in the interval [0, $2\bar{r}$] where $\bar{r}$ is the jobs' release time average. The maximum release time is derived as $2\bar{r} = 2\bar{p}\mu\rho/(2-\phi\mu\rho)$ where release time range factor (ρ) is $\rho = 2\bar{r}/\hat{C}_{max}$ and estimated makespan is $\hat{C}_{max} = (\phi\bar{r} + \bar{p}).\mu$ as defined earlier.

6. Due dates are calculated by $d_j = r_j + \alpha.p_j$ formula and deadlines are calculated by $D_j = d_j + \gamma.p_j$ formula where α and γ are the due date and d-to-D window tightness factors respectively as defined in Section 4.1.

7. New job arrival time (r_{new}) is assumed to be uniformly distributed in the early job arrival time range of $\left[0.1\times\hat{C}_{max},\ 0.3\times\hat{C}_{max}\right]$ and late job arrival time range of $\left(0.3\times\hat{C}_{max},\ 0.5\times\hat{C}_{max}\right]$

8. For low priority deviation level, the new weight (w_j') of randomly selected job is determined by uniformly distributed integers in the interval [w_j, 9]. The new due date (d_j') is calculated by $d_j' = r_j + \alpha'.p_j$ where α' is uniformly distributed in the interval [1.5, 2). The new deadline (D_j') is calculated by $D_j' = r_j + (\alpha' + \gamma').p_j$ where γ' is uniformly distributed in the interval [0, 1). High priority deviation level for urgent job is illustrated by setting $w_j' = 9$ which is the highest weight value, $d_j' = r_j + p_j$ where $\alpha' = 1$ which is the lowest possible value to complete processing, and $D_j' = d_j'$ where $\gamma' = 0$, the lowest value without cost tolerance.

9. Disrupted job is selected randomly among the jobs that have not been started by the time the disruption occurs for both priority and departure disruption. The same disruption times that were generated earlier by the early time range for job arrival disruption, are used for the latter disruptions to work on relatively large sets to reschedule. In the following sections, we address the different disruption events.

6.2 Effectiveness of the Algorithms for Small Size Problems

In order to apply different types of disruption events, five different instances are generated by using middle factor levels (μ=4, α=2, γ=1, ρ=0.2). The proposed algorithms were implemented in C++ and optimal solutions were obtained by implementing the MILP model (Section 5.2) in OPL Studio 6.3 with CPLEX 12.1 solver. Intel Core 2 Duo 2.10 GHz CPU with 2.00 GB of RAM was used to perform the computations.

The MILP model could be used for solving small size problems with up to 12 jobs in acceptable time. Thus, the performance of the algorithms are measured by computing the difference between the combined objective function value of the algorithm and the optimal solution value as follows:

$$\text{Relative Error} = \frac{f_{Algorithm} - f_{Optimal}}{f_{Optimal}} \tag{35}$$

The same ideal and nadir objective vectors ([f_{TWT}^{*} $f_{Instability}^{*}$] and [f_{TWT}^{max} $f_{Instability}^{max}$]) are used while calculating the objective function values for each algorithm.

6.2.1 Effectiveness of the Algorithms for Job Arrival

Rescheduling algorithms to repair job arrival disruptions are tested under various job arrival time ranges and objective weight coefficient, λ values. Job arrival time range which determines the set of jobs to reschedule can be an indicator of the rescheduling problem size. We assume that a new job arrive at a random time within this job arrival time range. All jobs that have not been started by the time this job arrive are considered in the rescheduling set. To have reasonable job arrivals, two time range levels (early, late) are defined. Thus, early job arrivals indicate large set of jobs to reschedule and vice versa for late job arrivals. The upper bound of job arrival time is compared to the estimated makespan ($\hat{C}_{max}$). Recall that the makespan (C_{max}) is the maximum completion time of all released jobs and is estimated depending on the schedule as discussed in the Subsection 6.2. The objective weight coefficient in the combined objective function (33) characterizes the impact of the relative importance weights on the problem objectives. In order to study its impact of the weight coefficient, three objective weight coefficient values (λ = 0.2, 0.5, 0.8) are used to obtain different pareto-optimal points. Moreover, since a little change in the instability component may result in high effect to the normalized objective function, the experiment is repeated by additional high objective weight coefficient values (λ = 0.92, 0.95, 0.98) to reduce this effect.

The performance values of the MetaRE, BestInSERT and SEPRE algorithms are compared in terms of relative error and computation time over levels of factors. Five different unique problem instances with the original schedule and job arrival time were

generated and these instances are used for different objective weight coefficient levels. To understand the results in Table 26, a sample batch of MetaRE solutions is given for early job arrival with λ= 0.5 in Table 25.

	λ= 0.5							
Job Arrival	Optimal			MetaRE			Relative Error	CPU(sec.)
	TWT	Instability	$f_{Optimal}$	TWT	Instability	f_{MetaRE}		
	289.99	1	0.2559	290.14	1	0.2561	0.00	0.399
	413.63	1	0.1385	413.63	1	0.1385	0.00	0.366
Early	458.51	1	0.1579	484.80	2	0.2670	0.69	0.464
	485.25	1	0.0997	485.25	1	0.0997	0.00	0.446
	634.50	2	0.2526	989.00	1	0.5012	0.98	0.500
						Average	0.34	0.435

Table 25 A sample experiment result for the MetaRE algorithm for early job arrival with λ= 0.5

λ	**0.2**			**0.5**			**0.8**		
Job Arrival	MetaRE	BestINSERT	SEPRE	MetaRE	BestINSERT	SEPRE	MetaRE	BestINSERT	SEPRE
Early	0.49	**0.00**	5.17	0.34	**0.26**	3.52	**0.31**	1.07	4.04
Late	25.02	**0.04**	54.19	6.92	**0.04**	13.79	3.49	**1.25**	4.98

λ	**0.92**			**0.95**			**0.98**		
Job Arrival	MetaRE	BestINSERT	SEPRE	MetaRE	BestINSERT	SEPRE	MetaRE	BestINSERT	SEPRE
Early	**0.41**	2.82	6.23	**0.54**	4.43	8.52	**1.05**	9.70	16.42
Late	4.88	**4.25**	5.00	7.07	7.36	**6.79**	17.29	19.87	**15.49**

Table 26 Average relative error of the algorithms for job arrival disruption

Prior to statistical analysis for the algorithm comparisons, a normality test should be performed to determine whether the data satisfies the normality assumption for parametric analysis methods. Probability plot and Kolmogorov-Smirnov normality test result obtained by Minitab 15.1 statistical software program for the relative error data are given in Figure 20 which shows that the relative error data for each combination does not follow a normal distribution ($p < 0.05$). Thus, the statistical significance of performance between algorithms can be analyzed by nonparametric Kruskal–Wallis test.

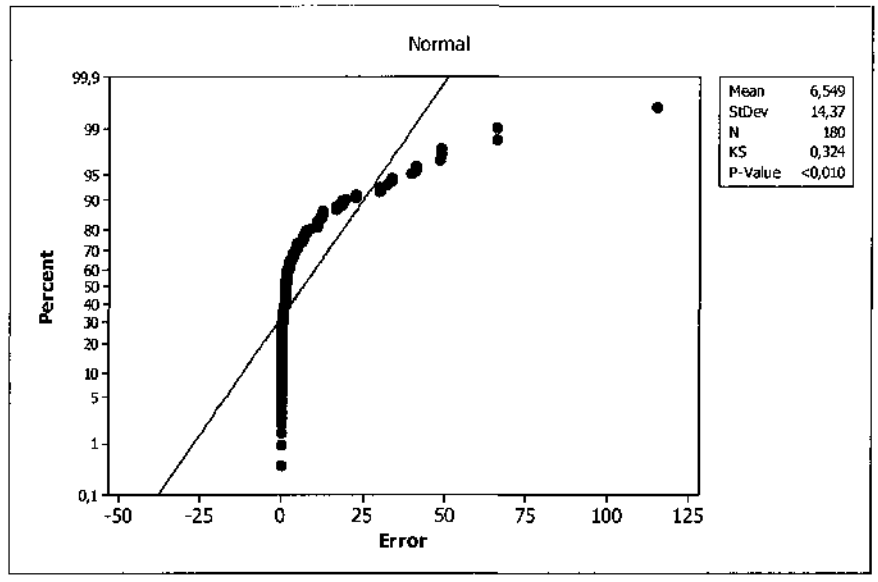


Figure 20 Normality Test for Relative Error Data

Nonparametric test is a hypothesis test that does not require the population's distribution to be characterized by certain parameters. They are useful when the data is strongly nonnormal and resistant to transformation. Kruskal-Wallis test is a nonparametric test that can be used whether two or more independent samples come from identical populations or not, and does not require the data to be normal. It instead uses the rank of the data values rather than the actual data values for the analysis. The Kruskal–Wallis test result (using Minitab 15.1 statistical software program) is given Table 27.

Algorithm	N	Median	Ave Rank	Z
BestINSERT	60	0.1605	70.7	-3.60
MetaRE	60	0.7175	80.3	-1.87
SEPRE	60	3.4473	120.5	5.47
Overall	180		90.5	

H = 30.90 DF = 2 P = 0.000
H = 31.20 DF = 2 P = 0.000 (adjusted for ties)

Table 27 Kruskal-Wallis Test on Relative Error of the algorithms for job arrival disruption

Table 27 shows that there is a statistical difference between the population mean values of performance in terms of relative error for the three algorithms ($p < \alpha=0.05$). However, the test does not reveal which means differ significantly. Thus, pair tests need to be performed and their results in Tables 28, Table 29 and Table 30 show that both BestINSERT and MetaRE algorithms have better mean performance than SEPRE algorithm ($p < 0.05$). However, it cannot be concluded that there exists significant difference between the BestINSERT and MetaRE algorithms ($p > 0.05$ in Table 28).

Algorithm	N	Median	Ave Rank	Z
BestINSERT	60	0.1605	57.2	-1.04
MetaRE	60	0.7175	63.8	1.04
Overall	120		60.5	

H = 1.09 DF = 1 P = 0.296
H = 1.13 DF = 1 P = 0.288 (adjusted for ties)

Table 28 Kruskal-Wallis Test on Relative Error of the BestINSERT and MetaRE algorithms

Algorithm	N	Median	Ave Rank	Z
BestINSERT	60	0.1605	44.0	-5.19
SEPRE	60	3.4473	77.0	5.19
Overall	120		60.5	

H = 26.89 DF = 1 P = 0.000
H = 27.06 DF = 1 P = 0.000 (adjusted for ties

Table 29 Kruskal-Wallis Test on Relative Error of the BestINSERT and SEPRE algorithms

Algorithm	N	Median	Ave Rank	Z
MetaRE	60	0.7175	46.9	-4.27
SEPRE	60	3.4473	74.1	4.27
Overall	120		60.5	

H = 18.25 DF = 1 P = 0.000
H = 18.30 DF = 1 P = 0.000 (adjusted for ties)

Table 30 Kruskal-Wallis Test on Relative Error of the MetaRE and SEPRE algorithms

A detailed analysis by using Kruskal-Wallis test for three algorithms is given in Appendix F. The results show that MetaRE and SEPRE perform better in terms of average relative error for early job arrivals, although BestINSERT has better performance for late job arrivals. Moreover, performance of BestINSERT declines as TWT objective weight increases. BestINSERT is significantly superior to the other algorithms when the instability objective is more important (for low λ values ≤ 0.5) than the TWT objective.

Table 31 shows that the average CPU times of the BestINSERT and SEPRE were less than 0.001 second which is much shorter than MetaRE's average CPU times for all cases. The average CPU time effectiveness of all algorithms does not change significantly with the change in λ values. The average CPU times of the MetaRE and BestINSERT clearly get shorter for late job arrival time range as it results in a smaller set of jobs to reschedule. Note that since the CPUs are in second, it does not take too long even for

λ	0.2			0.5			0.8		
Job Arrival	MetaRE	BestINSERT	SEPRE	MetaRE	BestINSERT	SEPRE	MetaRE	BestINSERT	SEPRE
Early	0.4266	0.0002	0.0000	0.4350	0.0004	0.0000	0.4360	0.0004	0.0002
Late	0.2158	0.0000	0.0002	0.2266	0.0002	0.0002	0.2240	0.0002	0.0002
λ	0.92			0.95			0.98		
Job Arrival	MetaRE	BestINSERT	SEPRE	MetaRE	BestINSERT	SEPRE	MetaRE	BestINSERT	SEPRE
Early	0.4324	0.0002	0.0004	0.4496	0.0002	0.0000	0.4314	0.0006	0.0000
Late	0.2284	0.0000	0.0002	0.2224	0.0004	0.0008	0.2222	0.0002	0.0000

Table 31 CPU times in sec. of the repair algorithms for job arrival disruption

MetaRE although it is obvious that MetaRE's average CPU times for all cases is longer than other algorithms. As a result, MetaRE and BestINSERT have significantly performed better than SEPRE in terms of average relative error. Especially, when the instability objective have a higher importance (for low λ values ≤ 0.5), BestINSERT is preferred.

6.2.2 Effectiveness of the Algorithms for Job Priority Deviation

Rescheduling algorithms to repair job priority disruptions are tested under various priority deviation levels and objective weight coefficient, λ values. Priority deviation level measures the amount of change in the urgency of a selected job. We defined only positive deviations for jobs by assuming that analysis for slight positive and negative deviations have similar results. Moreover, job cancellation disruption that will be tested separately, can be considered as a negative extreme change in the priority. Note that urgent job disruption is also a positive extreme change in the priority. Positive priority deviation is represented by increasing the weight (w_j) and tightening both due date (d_j) and d-to-D window for an arbitrary job. Two priority deviation levels (low, high) are defined by setting ranges for due date tightness (α) and d-to-D window tightness (γ) parameters. Note that a "High" level here means an urgent job disruption.

The performance values of the algorithms are compared in terms of relative error and computation time over different objective weight coefficient levels. The same five different unique problem instances which were generated earlier, are assumed to be affected by priority disruptions. Comparison results for job priority deviation algorithms are given in Table 32.

λ	**0.2**			**0.5**			**0.8**		
Priority Deviation	Shuffle	BestINSERT	MetaRE	Shuffle	BestINSERT	MetaRE	Shuffle	BestINSERT	MetaRE
Low	1.04	**0.44**	1.19	1.10	**0.29**	0.65	3.91	2.42	**0.62**
High	1.10	**0.82**	1.43	0.46	**0.15**	0.63	1.88	1.45	**0.61**

Table 32 Average relative error of the algorithms for job priority disruption

Algorithm	N	Median	Ave Rank	Z
BestINSERT	30	0.5540	40.7	-1.24
MetaRE	30	0.3829	42.2	-0.85
SHUFFLE	30	0.8596	53.6	2.09
Overall	90		45.5	

H = 4.41 DF = 2 P = 0.110
H = 4.42 DF = 2 P = 0.110 (adjusted for ties)

Table 33 Kruskal-Wallis Test on Relative Error of the algorithms for priority disruption

Since the relative error data for the priority disruption repair algorithms does not follow a normal distribution according to Kolmogorov-Smirnov normality test, the statistical significance of performance between algorithms are analyzed via using Kruskal–Wallis test. It can not be concluded that there exists significant difference between the algorithms ($p > 0.05$ in Table 33). Moreover, the detailed test results given in Appendix F shows that relative error performance of the SHUFFLE algorithm significantly declines as the TWT objective weight increases.

λ	0.2			0.5			0.8		
Priority Deviation	Shuffle	BestINSERT	MetaRE	Shuffle	BestINSERT	MetaRE	Shuffle	BestINSERT	MetaRE
Low	0.00	0.00	0.30	0.00	0.00	0.31	0.00	0.00	0.31
High	0.00	0.00	0.28	0.00	0.00	0.28	0.00	0.00	0.29

Table 34 CPU times of the repair algorithms for job priority disruption

Table 34 shows that MetaRE's average CPU time for all cases is reasonable, although it is clearly longer than the other algorithms. The average CPU time effectiveness of all algorithms does not change significantly between λ values and priority deviation levels.

6.2.3 Effectiveness of the Algorithms for Job Departure

Rescheduling algorithms to repair job departure disruptions are tested under various objective weight coefficient (λ) values. The same five instances and disruption times in the effectiveness test for job priority deviation are used to compare the algorithms for job departure disruption in terms of relative error and computational time over different objective weight coefficient levels. Comparison results for job departure repair algorithms are given in Table 35.

λ	0.2		0.4		0.5		0.6		0.8	
	LSHIFT	MetaRE	LSHIFT	MetaRE	LSHIFT	MetaRE	LSHIFT	MetaRE	LSHIFT	MetaRE
Avg. Relative	2.74	**0.63**	1.35	**0.25**	1.23	**0.14**	1.54	**0.12**	3.63	**0.23**
CPU(sec.)	0.00	0.28	0.00	0.28	0.00	0.28	0.00	0.28	0.00	0.28

Table 35 Performance of the algorithms for job departure disruption

Algorithm	N	Median	Ave Rank	Z
LeftSHIFT	25	1.468820	35.9	5.05
MetaRE	25	0.005682	15.1	-5.05
Overall	50		25.5	

H = 25.55 DF = 1 P = 0.000
H = 25.58 DF = 1 P = 0.000 (adjusted for ties)

Table 36 Kruskal-Wallis Test on Relative Error of the algorithms for job departure disruption

Table 35 and Table 36 show that MetaRE has significantly performed better than LSHIFT in terms of average relative error with a reasonable average CPU time.

6.3 Effectiveness of the Algorithms for Large Size Problems

It was clear that it is not possible to obtain optimal solutions in a reasonable time for large size problems (with 60 jobs and 10 machines) by running the MILP model. In this section, effectiveness of the two general MetaRE and BestINSERT algorithms (Table 24) that are distinguished with their good performances in most job disruption cases, were compared to each other only for priority disruption type. Job arrival and job departure disruptions can be defined as a priority disruption. As was mentioned earlier in Section 6.2.2, job departure disruption can be considered as a negative extreme change in the priority of an existing job, while job arrival disruption can be considered as a positive extreme change in the priority of the additional job. In order to implement the algorithms, five different instances are generated by using factor levels (μ=6, α=2, γ=1, ρ=0.2). The proposed algorithms were implemented in C++ and Intel Core 2 Duo 2.10 GHz CPU with 2.00 GB of RAM was used to perform the computations.

The performance of the algorithms are measured by computing the difference between the combined objective function value of a particular algorithm and the minimum value of the algorithms as follows:

$$\text{Relative Difference} = \frac{f_{Algorithm} - \min(f_{MetaRE}, f_{BestINSERT})}{\min(f_{MetaRE}, f_{BestINSERT})} \quad (36)$$

The pessimistic and optimistic objective values are used as explained in Section 5.2 for the ideal and nadir objective vectors ($[f_{TWT}^{*} f_{Instability}^{*}]$ and $[f_{TWT}^{max} f_{Instability}^{max}]$) while calculating objective function values for each of the algorithms.

Two time range levels (early and late disruption) are defined like job arrival case to test the performance of the algorithms for different problem sizes. In order to study the impact of the weight coefficient, five objective weight coefficient values (λ = 0.1, 0.3, 0.5, 0.7, 0.9) are used to obtain different pareto-optimal points.

The performance values of the MetaRE and BestInSERT algorithms are compared in terms of relative error and computation time over levels of factors. Five different unique problem instances with the original schedule and disruption time were generated and these instances were used with different objective weight coefficient levels. The original schedules are obtained by selecting the best solution of 10 replicate (runs) of MetaRaPS algorithm that had the best performance for the scheduling problem. The results obtained by implementing the two algorithms are summarized in Table 37 The randomized MetaRE algorithm was replicated 10 times and the average was used to compare.

		BestINSERT		MetaRE	
Disruption	Weight	Rel.Diff.	CPU(sec.)	Rel.Diff.	CPU(sec.)
Early	0.1	0.55	0.00	0.00	4.66
	0.3	0.48	0.00	0.00	4.64
	0.5	0.21	0.00	0.00	4.60
	0.7	0.01	0.00	0.09	4.57
	0.9	0.00	0.00	0.61	4.67
Late	0.1	0.40	0.00	0.00	2.47
	0.3	0.42	0.00	0.00	2.49
	0.5	0.45	0.00	0.00	2.42
	0.7	0.19	0.00	0.00	2.40
	0.9	0.00	0.00	0.35	2.47
Overall		0.27	0.00	0.10	3.54

Table 37 Performance of the algorithms for priority disruption

Algorithm	N	Median	Ave Rank	Z
BestINSERT	50	0.279798125	61.7	3.86
MetaRE	50	0.000000000	39.3	-3.86
Overall	100		50.5	

H = 14.90 DF = 1 P = 0.000
H = 17.03 DF = 1 P = 0.000 (adjusted for ties)

Table 38 Kruskal-Wallis Test on Relative Difference for priority disruption (n=60)

Table 37 and Table 38 show that the MetaRE algorithm is significantly superior to the BestINSERT algorithm in terms of both relative error and computational time. The detailed test results are given in Appendix F. The relative performance of the BestINSERT algorithm gets better as the TWT objective weight increases.

CHAPTER 7

CONCLUSIONS AND FUTURE RESEARCH

7.1 Conclusions

This study addressed the parallel machine scheduling (ARSP) and rescheduling problem (ARRP) in a practical operational context requiring high quality solutions in an acceptable timeframe. ARSP was modeled as parallel machine scheduling problem with due date-to-deadline (d-to-D) windows and dynamic ready times to minimize total weighted tardiness. A piecewise tardiness cost was defined by taking into account d-to-D windows and job priorities to evaluate the performance of the schedule. First, a mixed integer linear programming model (MILP) was developed to find optimal solutions for ARSP. As the dimensions of the problem get larger, the solution process of mathematical modeling loses its effectiveness. Since ARSP is NP-hard, it is required to develop qualified, easily, and quickly implemented solution approaches with reasonable computation times.

A new composite dispatching rule called APTCR was developed to obtain good solutions quickly and its effectiveness was studied by comparing it with SA_{random} which is a Simulated Annealing algorithm that starts with a random solution. APTCR rule is an extension of the ATC rule but with considering d-to-D window and dynamic ready times. In addition to APTCR and SA_{random}, SA_{APTCR} and MetaRaPS were developed and implemented by integrating APTCR to construct initial solutions. The ARRP was also considered with three cases of job related disruptions (arrival of new job, departure of existing job, and changes in job priorities). The MILP model for ARSP was revised to take into account schedule instability by incorporating a new instability component into the objective function and adding new decision variables, parameters and constraints. Five regeneration and partial repair algorithms (MetaRE, BestINSERT, SEPRE, LSHIFT and SHUFFLE) were developed to update instantly the current schedule at the disruption time.

Extensive experiments were designed and conducted to study the effectiveness of the scheduling and rescheduling algorithms, which both were compared to optimal solutions for small size problems (n=12) and to each other for large size problems (n=60).

Computational experiments for ARSP demonstrate that although APTCR has significantly worse deviation performance from optimal solution than SA_{random} for small size problems, it is more likely to outperform SA_{random} when the problem size increases. Moreover CPU time of APTCR is significantly shorter than SA_{random} in both cases.MetaRaPS outperformed SA_{APTCR} in terms of average error from optimal solution for both small and large size problems; however, in terms of CPU time performance, SA_{APTCR} was significantly better than MetaRaPS in both cases. Results for small size problems shows that MetaRaPS algorithm is more robust compared to SA_{APTCR}.

It is observed that metaheuristics that are combined with the proposed APTCR algorithm, clearly resulted in better solutions than the APTCR algorithm and SA individually. Using only composite dispatching rules, such as the APTCR rule, does not produce the best solutions for most applications. On the other hand, starting initially from a better than random solution was advantageous in ARSP for which good solutions cannot be easily obtained. SA and MetaRaPS are promising metaheuristics with their simplicity and effectiveness to find high quality solutions for ARSP. and potentially for other similar scheduling problems.

Experiments were conducted separately for three disruption cases of ARRP with various values of objective weighting factor (λ) in extended analysis. First, rescheduling algorithms to repair job arrival disruptions were tested under various job arrival time ranges. MetaRE and BestINSERT have significantly performed better than SEPRE in terms of average relative errorwith no significant difference between the BestINSERT and MetaRE algorithms for small size problems. However, when the instability objective have a higher importance weight ($\lambda \leq 0.5$), BestINSERT can be preferred because the performance of BestINSERT declines as TWT objective weight increases. The average CPU time effectiveness of all algorithms does not change significantly with the change in λ values. Second, rescheduling algorithms to repair job priority disruptions were tested under various priority deviation levels. Job cancellation disruption can be considered as a

negative extreme change in the priority as well as urgent job disruption is a positive extreme change in the priority. There does not exist significant difference between the performance of the algorithms. Relative error performance of the SHUFFLE algorithm significantly declines as the TWT objective weight increases. MetaRE's average CPU time for all cases is reasonable, although it is clearly longer than the other algorithms. The average CPU time effectiveness of all algorithms does not change significantly for different λ values and priority deviation levels. Third, MetaRE has significantly performed better than LSHIFT to repair job departure disruptions in terms of average relative error with a reasonable average CPU time.

Finally, the effectiveness of the best performing algorithms MetaRE and BestINSERT in most job disruption cases were compared to each other for priority disruption type. MetaRE is significantly superior to the BestINSERT algorithm in terms of both relative error and computational time. The relative performance of the BestINSERT algorithm enhanced as the TWT objective weight increases.

7.2 Contributions

This dissertation research has the following intellectual and practical contributions:

1. ARSP under in-theater air force military environment was defined and an optimization model was formulated to solve the problem for the first time. The problem was formulated as an identical parallel machine scheduling problem which is commonly researched in production environments. This research provides highly valuable practical contributions for military operations as there is very little existing research on the ARSP for both inter-theater and in-theater operations. Effective AR scheduling affects the performance of air force in terms of responsiveness, endurance, flexibility, efficiency, and safety. Moreover, the study of fighter aircraft AR, provides basic knowledge for programming autonomous AR of unmanned aerial vehicles (UAV).

2. The problem characteristics make the research remarkable in the *scheduling theory* applications also. Multi-resourced, time window, dynamic, and multi objective perspectives of the problem qualify the study as a state-of-art research in

the scheduling discipline. Time window between due date and deadline were firstly considered in parallel machine scheduling problem by defining a piecewise tardiness cost. There is very little existing research addressing parallel machine scheduling and rescheduling with release time and due date-to-deadline windows to minimize the total weighted tardiness. The scheduling and rescheduling problem framed in this dissertation has never been tackled in the Operations Research literature. New Mixed Integer Programming models were introduced to find optimal solutions for the problem.

3. Approximate heuristic and metaheuristic algorithms were developed to obtain best schedules for the problem in a reasonable time. The newly proposed APTCR algorithm was used to find good initial solutions to be improved by other metaheuristics (SA and MetaRaPS) and also to repair the disrupted schedules in the rescheduling algorithms.

4. A multi objective optimization (MOO) approach was developed to update the disrupted schedules satisfactorily with respect to both objectives, and the trade-off between the objectives was effectively evaluated. New rescheduling algorithms (MetaRE, BestINSERT, SEPRE, LSHIFT, and SHUFFLE) were developed where MetaRaPS was firstly applied to solve the MOO problems.

5. The piecewise cost function used in this research has nonmilitary important applications such as capturing the tolerance of costumers to order delays and stockout costs. Similarly, time constraints, new rush orders, unexpected cancellations, multi objectives of marketing-production departments or changing market strategies cause scheduling complexities in the manufacturing system design. This study provides a system view of manufacturing including order ready times, order priority changes, production control systems with time windows, and machine utilization. This research stimulates further development and implementation of rescheduling models to mitigate the effects of disruptions that occur frequently in manufacturing environments.

7.3 Future Research

The problem can be extended in the future to address the following aspects:

1. *Stochastic or fuzzy variables* as a result of uncertain processing times, ready times, due dates, deadlines, and priorities.

2. *Multi objective* scheduling problems as a result of objectives related to efficiency of fuel transfer and total waiting time of the jobs.

3. *Rescheduling* extension as a result of machine availability problems caused by emergent events, machine availability constraints due to breakdowns, maintenance, or having a variable number of tankers.

4. More constraints such as machine compatibility, sequence dependent setup times and unexpected disruptions may be included in the models and algorithms. Modeling with (i) new *processing time* definition which is dependent on scheduled refueling start times, (ii) *compatibility* constraints due to having different types of tankers, (iii) treating fighters, instead of wings, as jobs and (iv) incorporating *communication delay* constraints.

5. This military application can be extended to manufacturing and other environments such as project scheduling, designing markets by considering customer expectations, satellite maintenance planning, programming autonomous AR of UAVs and military or civilian air traffic control.

REFERENCES

1. Abumaizar, R.J., & Svestka, J.A. (1997). Rescheduling job shops under random disruptions. *International Journal of Production Research*, 35, 2065–2082.

2. AFPAM 10-1403, Secretary of the Air Force, *Airlift Planning Factors*. (2003). *http://www.e-publishing.af.mil/.*

3. Alagoz, O., & Azizoglu, M. (2003). Rescheduling of identical parallel machines under machine eligibility constraints. *European Journal of Operational Research*, 149, 523–532.

4. Akturk, M.S., & Gorgulu, E. (1999). Match-up scheduling under a machine breakdown. *Journal of Operational Research*, 112, 81–97.

5. Arcus, A. L. (1966). COMSOAL: a computer method of sequencing operations for assembly lines. *International Journal of Production Research*, 4, 259–277.

6. Arnaout, J.P., & Rabadi G. (2008). Rescheduling of unrelated parallel machines under machine breakdowns. *International Journal of Applied Management Science*, *1*(1).

7. ATP-3.3.4.2 (ATP-56(B) Change 1), NATO STANAG 3971, *Air to Air Refueling*. (2008). *http://www.raf.mod.uk/downloads/airtoair56b.cfm*

8. Aytug, H., Lawley, M.A., McKay, K., Mohan, S., & Uzsoy, R. (2005). Executing production schedules in the face of uncertainties: a review and some future directions. *European Journal of Operational Research*, 161, 86–110.

9. Azizoglu, M., & Alagoz, O. (2005). Parallel-machine rescheduling with machine disruptions. *IIE Transactions*, 37, 1113–1118.

10. Barnes, J. W., Wiley, V. D., Moore J. T., & Ryer, D. M. (2004). Solving the Aerial Fleet Refueling Problem using Group Theoretic Tabu Search. *Mathematica and Computer Modeling*, 39, 617-640.

11. Bean, J. C., Birge, J. R., Mittenthal, J., & Noon, C. E. (1991). Match-up scheduling with multiple resources, release dates and disruptions. *Operations Research, 39*(3), 470-483.

12. Biskup D., Herrmann, J., & Gupta, J.N.D. (2008). Scheduling identical parallel machines to minimize total tardiness. *International Journal of Production Economics*, 115, 134–142.

13. Blazewicz, J., Ecker K. H., Pesch E., Schmidt G., & Weglarz J. (2007). *Handbook on Scheduling: From Theory to Applications*. Springer, New York.

14. Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: overview and conceptual comparison. *ACM Computer Surveys*, 35, 268–308.

15. Bolkcom, C. (2006). *Air Force Aerial Refueling Methods: Flying Boom versus Hose-and-Drogue.* CRS Report for Congress, RL32910. *http://www.fas.org/sgp/crs/weapons/RL32910.pdf.*

16. Cheng, H.-C., Chiang, T.-C., & Fu, L.-C. (2008). A Memetic algorithm for parallel batch machine scheduling with incompatible job families and dynamic job arrivals. *IEEE International Conference on Systems, Man and Cybernetics.*

17. Cheng, S-C., Shiau, D.-F., Huang, Y.-M., & Lin, Y.-T. (2009). Dynamic hard-real-time scheduling using genetic algorithm for multiprocessor task with resource and timing constraints. *Expert Systems with Applications*, 36, 852–860.

18. Church, L.K., & Uzsoy, R. (1992). Analysis of periodic and event driven rescheduling policies in dynamic shops. *International Journal of Computer Integrated Manufacturing*, 5, 153–163.

19. Curry, J., & Peters, B. (2005). Rescheduling parallel machines with stepwise increasing tardiness and machine assignment stability objectives. *International Journal of Production Research, 43*(15), 3231 – 3246.

20. Della Croce, F., Tadei, R., & Volta, G. (1995). A genetic algorithm for the job shop problem. *Computers and Operations Research*, 22, 15 – 24.

21. DePuy G. W., Whitehouse G. E., & Moraga R. J. (2001). MetaRaPS: A simple and efficient approach for solving combinatorial problems. *29th International Conference on Computers and Industrial Engineering*, November 1-3, Montreal, Canada, 644-649.

22. DePuy, G. W., Moraga, R. J., & Whitehouse, G. E. (2005). MetaRaPS: a simple and effective approach for solving the traveling salesman problem. *Transportation Research Part E: Logistics and Transportation Review*, *41*(2), 115–130.

23. Dorigo M., & Gambardella, L.M. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions in Evolutionary Computation*, 1, 53–66.

24. Dorigo M., & Stützle T. (2005). *Ant Colony Optimization*. Prentice-Hall of India Pvt. Ltd.

25. Driessel, R., & Mönch, L. (2009). Scheduling jobs on parallel machines with sequence-dependent setup times, precedence constraints, and ready times using variable neighborhood search. *Proceedings of the 2009 IEEE International Conference on Industrial Engineering and Engineering Management.*

26. Duenas, A., & Petrovic, D. (2008). An approach to predictive-reactive scheduling of parallel machines subject to disruptions. *Annual Operations Research*, 159, 65–82.

27. Eom, D.-H., Shin, H.-J., Kwun, I.-H., Shim, J.-K., & Kim, S.-S. (2002). Scheduling jobs on parallel machines with sequence-dependent family set-up times. *International Journal of Advanced Manufacturing Technology*, 19, 926–932.

28. Feo, T., & Resende, M. (1995) Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6, 109–133.

29. GAO/NSIAD-94-68. (1993). *An Assessment of Aerial Refueling Operational Efficiency Operation Desert Storm.* http://www.*archive.gao.gov/t2pbat5/150241.pdf*

30. Garey, M.R., & Johnson, D.S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco: W. H. Freeman and Company.

31. Gharehgozli, A.H., Tavakkoli, R., & Zaerpour, N. (2009). A fuzzy-mixed-integer goal programming model for a parallel-machine scheduling problem with sequence-dependent setup times and release dates. *Robotics and Computer-Integrated Manufacturing*, 25, 853–859.

32. Grodzevich, O., & Romanko, O. (2006). Normalization and other topics in multi-objective optimization. *Proceedings of the Fields–MITACS Industrial Problems Workshop.*

33. Hall, N.G., & Potts, C.N. (2004). Rescheduling for new orders. *Operations Research*, 52, 440–453.

34. Harel, D. and Feldman, Y. (2004). *Algorithmics : The Spirit of Computing*. 3rd Ed., Addison-Wesley Publishers Ltd., Essex.

35. Hasija, S., & Rajendran, C. (2004). Scheduling in flowshops to minimize total tardiness of jobs. *International Journal of Production Research*, 42, 2289–301.

36. Hazır, Ö., Günalay, Y., & Erel, E. (2008). Customer order scheduling problem: a comparative metaheuristics study. *International Journal of Advanced Manufacturing Technology*, 37, 589–598.

37. Hepdogan S., Moraga R., DePuy G.W., & Whitehouse G.E. (2009). A MetaRaPS for the early/tardy single machine scheduling problem. *International Journal of Production Research, 47*(7), 1717–1732.

38. Hoitomt, D. J., Luh, P. B., Max, E., & Pattipati, K. R. (1990). Scheduling jobs with simple precedence constraints on parallel machines. *IEEE Control Systems Magazine.*

39. Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, MI.

40. Itayef, A. B., Loukil, T., & Teghem, J. (2009). Rescheduling a permutation flowshop problems under the arrival a new set of jobs. *IEEE Transactions.*

41. Jain, A.K., & ElMaraghy, H. (1997). Production scheduling/rescheduling in flexible manufacturing. *International Journal of Production Research*, 35, 281–309.

42. Jain, A.S., & Meeran, S. (1998). Job-shop scheduling using neural networks. *International Journal of Production Research*, 36, 1249–1272.

43. Jin, Z., Shima, T., & Schumacher, C. J. (2006). Optimal scheduling for refueling multiple autonomous aerial vehicles. *IEEE Transactions on Robotics*, *22*(4).

44. Jozefowska, J., Mika, M., Rozycki, R., Waligora, G., and Weiglarz, J., European Journal of Operational Research, 107, 354-370, 1998.

45. Jungwattanakit, J., Reodechaa, M., Chaovalitwongsea, P., & Werner, F. (2009). A comparison of scheduling algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria, *Computers & Operations Research,* 36, 358 – 378.

46. Karp., R.M. (1972). Reducibility among combinatorial problems. In R.E. Miller, & J.W. Tatcher (Eds.), *Complexity of Computer Computations*, 85-103, New York: Plenum Press, 1972.

47. Kim,Y.D., Lim, H.G., & Park, M.W. (1996). Search heuristics for a flowshop scheduling problem in a printed circuit board assembly process. *European Journal of Operational Research*, 91, 124–43.

48. Lan, G., DePuy, G. W., & Whitehouse, G. E. (2007). An effective and simple metaheuristic heuristic for the set covering problem. *European Journal of Operational Research,* 176, 1387-1403.

49. Lee, C-Y., Lei, L., & Pinedo, M. (1997). Current trends in deterministic scheduling. *Annals of Operations Research*, 70, 1 – 41.

50. Lee, C.-Y., Leung, J. Y-T., & Yu, G. (2006). Two machine scheduling under disruptions with transportation considerations. *Journal of Scheduling*, 9, 35–48.

51. Lee, Y.H., & Pinedo, M. (1997). Theory and methodology: Scheduling jobs on parallel machines with sequence-dependent setup times. *European Journal of Operational Research*, 100, 464-474.

52. Li, H., Li, Z., Li, L. X., & Hu, B. (2000). A production rescheduling expert simulation system. *European Journal of Operational Research*, 124, 283–293.

53. Liaw C.-F., Lin, Y.-K., Cheng, C.Y., & Chen, M. (2003). Scheduling unrelated parallel machines to minimize total weighted tardiness. *Computers & Operations Research*, 30, 1777–1789.

54. Logendran, R., & Subur, F. (2004). Unrelated parallel machine scheduling with job splitting. *IIE Transactions*, 36, 359–372.

55. Marler, R.T., & Arora, J.S. (2004). Survey of multi-objective optimization methods for engineering. Structural Multidisciplinary Optimization, 26, 369–395.

56. Marler, R.T., & Arora, J.S. (2005). Function-transformation methods for multi-objective optimization. *Engineering Optimization*, *37*(6), 551–570.

57. Mason, S.J., Jin, S., & Wessels, C.M. (2004). Rescheduling strategies for minimising total weighted tardiness in complex job shops. *International Journal of Production Research*, 42, 613–628.

58. McKay K. N., Morton T.E., Ramnath P., & Wang J. (2000). 'Aversion dynamics' scheduling when the system changes. *Journal of Scheduling*, *3*(71), 88.

59. Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., & Teller, E. (1956). Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21, 1087-1092.

60. Miettinen, K. (1999). *Nonlinear Multi-objective Optimization*, Kluwer Academic Publishers: Boston.

61. Mokotoff, E. (2001). Parallel machine scheduling problems: A survey. *Asia - Pacific Journal of Operational Research, 18*(2), 193.

62. Montgomery, D.C. (2001). *Design and Analysis of Experiments*, Wiley, New York, NY.

63. Moraga R. J., DePuy G. W., & Whitehouse G. E. (2006). *Metaheuristics: A Solution Methodology for Optimization Problems, Handbook of Industrial and Systems Engineering*. CRC Press, FL.

64. Morton, T., & Pentico, D. (1993). *Heuristic scheduling systems: With applications to production systems and project management.* New York: John Wiley and Sons.

65. Mönch, L. (2008). Heuristics to minimize total weighted tardiness of jobs on unrelated parallel machines, *4th IEEE Conference on Automation Science and Engineering*, Key Bridge Marriott, Washington DC, USA, August 23-26.

66. Mönch, L., Balasubramanian, H., Fowler, J.W., & Pfund, M.E. (2005). Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times. *Computers and Operations Research*, 32, 2731–2750.

67. Mönch, L., Zimmermann, J., & Otto, P. (2006). Machine learning techniques for scheduling jobs with incompatible families and unequal ready times on parallel batch machines. *Engineering Applications of Artificial Intelligence*, 19, 235–245.

68. Parthasarathy, S., & Rajendran, C. (1998). Scheduling to minimize mean tardiness and weighted mean tardiness in flowshop and flowline-based manufacturing cell. *Computers and Industrial Engineering*, 34, 531–546.

69. Paula M.R., Mateus, G.R., & Ravetti, M.G. (2010). A non-delayed relax-and-cut algorithm for scheduling problems with parallel machines, due dates and

sequence-dependent setup times. *Computers & Operations Research*, 37, 938 - 949.

70. Pfund, M., Fowler, J. W., Gadkari, A., & Chen, Y. (2008). Scheduling jobs on parallel machines with setup times and ready times. *Computers and Industrial Engineering*, 54, 764–782.

71. Phadke, S. M. (1989). *Quality Engineering Using Robust Design*, Prentice Hall, Englewood Cliffs.

72. Pinedo, M. (2008). *Scheduling Theory, Algorithms, and Systems*. 3rd Ed., Springer, New York.

73. Pirlot, M. (1996). General local search methods. *European Journal of Operational Research*, 92, 493-511.

74. Rabadi G., Moraga R.J., & Al-Salem A. (2006). Heuristics for the unrelated parallel machine scheduling problem with setup times. *Journal of Intelligent Manufacturing*, 17, 85–97.

75. Rachamadugu, R. V., & Morton, T. E. (1982). Myopic heuristics for the single machine weighted tardiness problem. Working Paper, 30, 82–83, GSIA. Pittsburgh, PA: Carnegie Mellon University.

76. Raghavan, N. R. S., & Venkataramana, M. (2009). Parallel processor scheduling for minimizing total weighted tardiness using ant colony optimization. *International Journal of Advanced Manufacturing Technology*, 41, 986–996.

77. Raheja, S.A., & Subramaniam, V. (2002). Reactive recovery of job shop schedules- A review. *International Journal of Advanced Manufacturing Technology*, 19, 756–763.

78. Rangaiah, G. P. (2008). *Multi-Objective Optimization: Techniques and Applications in Chemical Engineering*. World Scientific Publishing.

79. Reichelt, D., Mönch, L., Gottlieb, J. & Raidl, G.R. (2006). Multiobjective scheduling of jobs with incompatible families on parallel batch machines, *EvoCOP 2006*, LNCS 3906, Berlin Springer-Verlag Heidelberg, 209–221.

80. Sanchez, S. M. (2008), Better than a petaflop: the power of efficient experimental design, Proceedings of the 2008 Winter Simulation Conference, S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler eds.

81. Shi-jin, W., Li-feng, X. & Bing-hai, Z. (2007). Filtered-beam-search-based algorithm for dynamic rescheduling in FMS. *Robotics and Computer-Integrated Manufacturing*, 23, 457–468.

82. Silver, E. (2002). *An Overview of Heuristic Solution Methods*, Haskayne School of Business University of Calgary, Working Paper – 2002-15, October.

83. Simpson, T. W., Peplinski, J. D., Koch, P. N., & Allen J. K. (1997). On the use of statistics in design and the implications for deterministic computer experiments, Proceedings of ASME Design Engineering Technical Conferences, Sacramento, California.

84. Subramaniam, V. & Raheja, A.S. (2003). mAOR: a heuristic-based reactive repair mechanism for job shop schedules. *International Journal of Advanced Manufacturing Technology*, 22, 669–680.

85. Subramaniam, V., Raheja, A.S. & Reddy, K.R.B. (2005). Reactive repair tool for job shop schedules. *International Journal of Production Research*, 43, 1–23.

86. Sun, J,. and Xue, D. (2001). A dynamic reactive scheduling mechanism for responding to the changes of production orders and manufacturing resources. Computers in Industry, 46, 189–207.

87. Tan, K. C., Lee, L. H., Zhu, Q. L., Ou, K. (2001). Heuristic methods for vehicle routing problem with time windows. *Artificial Intelligence in Engineering*, 15, 281-295.

88. Tanaka, S., & Araki, M. (2008). A branch-and-bound algorithm with Lagrangian relaxation to minimize total tardiness on identical parallel machines. *International Journal Production Economics*, 113, 446–458.

89. Tang, L., Wang, G., & Liu, J. (2007). A branch-and-price algorithm to solve the molten iron allocation problem in iron and steel industry. *Computers & Operations Research*, 34, 3001 – 3015.

90. Tavakkoli-Moghaddam, R., Taheri, F., Bazzazi M., Izadi M., & Sassani, F. (2009). Design of a genetic algorithm for bi-objective unrelated parallel machines scheduling with sequence-dependent setup times and precedence constraints. *Computers & Operations Research*, 36, 3224 – 3230.

91. Unal, A.T., Uzsoy, R., & Kiran, A.S. (1997). Rescheduling on a single machine with part-type dependant setup times and deadlines. *Annals of Operations Research*, 70, 93–113.

92. Vallada, E. Ruiz, R, & Minella, G. (2008). Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics. *Computers and Operations Research*, 35, 1350 – 1373.

93. Vepsalainen, A., & Morton, T. (1987). Priority rules for job shops with weighted tardiness costs. *Management Science*, 33, 1035–1047.

94. Vieira, G.E., Herrmann, J. W., & Lin E. (2000). Predicting the performance of rescheduling strategies for parallel machine systems. *Journal of Manufacturing Systems*, *9*(4), 256-266.

95. Vieira, G.E., Herrmann, J. W., & Lin E. (2003). Rescheduling manufacturing systems: a framework of strategies, policies and methods. *Journal of Scheduling*, *6*(39), 62.

96. Wu, S. D., R. H. Storer, P.-C. Chang. (1993). One-machine rescheduling heuristics with efficiency and stability as criteria. *Computers and Operations Research*, 20, 1-14.

97. Yang, B. (2007). Single machine rescheduling with new jobs arrivals and processing time compression. *International Journal of Advanced Manufacturing Technology*, 34, 378–384.

98. Yang, M.-H., Chung, S.-H., & Kao, C.-K. (2006). A comparative study to minimize the makespan of parallel-machine problem with job arrival in uncertainty. *IIE Conference Proceedings*.

99. Yang, R.J., Tseng, L., & Cheng, J. (1994). Feasibility study of crash optimization. *Advances in Design Automation, Proceedings of the 1994ASME Design Technical Conference*, Minneapolis, 2, 549–556.

100. Zobolas, G.I., Tarantilis, C.D., & Ioannou, G. (2008). Exact, heuristic and metaheuristic algorithms for solving shop scheduling problems. *Studies in Computational Intelligence (SCI)*, 128, 1–40.

APPENDICES

APPENDIX A: SAMPLE DATA FOR ARSP AND ARRP

Problem Factors

μ	α	γ	ρ	k1	k2	k3
4.0	2.0	1.0	0.2	2.168	0.980	0.190

Scheduling Problem

Pjobs = {0,1,2,3,4,5,6,7,8,9,10,11,12};
Sjobs = {1,2,3,4,5,6,7,8,9,10,11,12};
Machines = {1,2,3};
Release = [4.67,4.17,18.67,6.00,18.83,3.83,10.50,11.33,3.50,16.33,19.00,5.33];
Processing Time = [40, 15, 16, 24, 15, 16, 22, 39, 20, 41, 22, 36];
Weight = [9, 7, 1, 9, 2, 5, 1, 2, 5, 4, 5, 8];
DueDate = [84.67,34.17,50.67,54.00,48.83,35.83,54.50,89.33,43.50,98.33,63.00,77.33];
Deadline = [124.67,49.17,66.67,78.00,63.83,51.83,76.50,128.33,63.50,139.33,85.00,113.33];
FixedCost = 120;

Job Arrival Disruption Rescheduling Problem

r_{new} = 26.00,
p_{new}= 30, w_{new}= 5, d_{new}= 86, D_{new}= 116

Pjobs = {0,1,2,3,4,5,6,7,8,9,10};
Sjobs = {1,2,3,4,5,6,7,8,9,10};
Machines = {1,2,3};
Release = [26, 4.67,18.67,0,0,10.50,11.33,16.33,0,5.33];
ProcessingTime = [30, 40, 16, 47.5, 34.17, 22, 39, 41, 41.83, 36];
Weight = [5, 9, 1, 999, 999, 1, 2, 4, 999, 8];
DueDate = [86, 84.67,50.67,47.5,34.17,54.50,89.33,98.33,41.83,77.33];
Deadline = [116, 124.67,66.67,47.5,34.17,76.50,128.33,139.33,41.83,113.33];
FixedCost = 120;

fmin = [284.33 0];
fmax = [761.5 2];
r= 0.92;
OldCompTime = [0 87.5 50.17 47.5 34.17 72.17 116.83 113.17 41.83 77.83];
OldAssign = [[0 0 0] [1 0 0] [0 0 1] [1 0 0] [0 0 1] [0 0 1] [0 1 0] [0 0 1] [0 1 0] [0 1 0]];

Job Departure Disruption Rescheduling Problem

Disruption decision time = 26.00
Job index = 7, machine index = 1;

Pjobs = {0,1,2,3,4,5,6,7,8};
Sjobs = {1,2,3,4,5,6,7,8};
Machines = {1,2,3};
Release = [4.67,18.67,0,0,10.50, 16.33,0,5.33];
ProcessingTime = [40, 16, 47.5, 34.17, 22, 41, 41.83, 36];
Weight = [9, 1, 999, 999, 1, 4, 999, 8];
DueDate = [84.67,50.67,47.5,34.17,54.50, 98.33,41.83,77.33];
Deadline = [124.67,66.67,47.5,34.17,76.50, 139.33,41.83,113.33];
FixedCost = 120;

fmin = [0 0];
fmax = [200 1];
r= 0.2;
OldCompTime = [87.5 50.17 47.5 34.17 72.17 113.17 41.83 77.83];
OldAssign = [[1 0 0] [0 0 1] [1 0 0] [0 0 1] [0 0 1] [0 0 1] [0 1 0] [0 1 0]];

Priority Disruption Rescheduling Problem

Disruption decision time = 26.00
job index = 7, machine index = 1
$w_{old} = 2$, $d_{old} = 89.3$, $D_{old} = 128.3$
$W_{new} = 5$, $d_{new} = 75.5$, $D_{new} = 84.9$

Pjobs = {0,1,2,3,4,5,6,7,8,9};
Sjobs = {1,2,3,4,5,6,7,8,9};
Machines = {1,2,3};
Release = [4.67,18.67,0,0,10.50,11.33,16.33,0,5.33];
ProcessingTime = [40, 16, 47.5, 34.17, 22, 39, 41, 41.83, 36];
Weight = [9, 1, 999, 999, 1, 5, 4, 999, 8];
DueDate = [84.67,50.67,47.5,34.17,54.50,75.4684,98.33,41.83,77.33];
Deadline = [124.67,66.67,47.5,34.17,76.50,84.8867,139.33,41.83,113.33];
FixedCost = 120;

fmin = [332.83 1];
fmax = [706.5 4];
r= 0.2;
OldCompTime = [87.5 50.17 47.5 34.17 72.17 116.83 113.17 41.83 77.83];
OldAssign = [[1 0 0] [0 0 1] [1 0 0] [0 0 1] [0 0 1] [0 1 0] [0 0 1] [0 1 0] [0 1 0]];

APPENDIX B : RESULTS FOR PARAMETER SETTING FOR ARSP'S ALGORITHMS

Table 1 Experiment Results for Parameter Setting of the APTCR Rule

Instance	μ	α	γ	ρ	k1	k2	k3
1	1	1	1	1	2 2426	2 2426	3 7952
2	1	1	1	0	1 9532	1 9532	3 5456
3	1	1	1	1	2 0572	2 0572	1 6796
4	1	1	0	1	3 1682	0 9058	2 1224
5	1	1	0	0	2 7928	0 5734	0 3772
6	1	1	0	1	2 0518	0 476	0 2506
7	1	1	1	1	2 6914	1 8274	1 2996
8	1	1	1	0	1 8384	2 3332	0 9884
9	1	1	1	1	1 7904	3 3806	0 9122
10	1	0	1	1	1 8754	1 8754	0 9332
11	1	0	1	0	2 0538	2 0538	0 2864
12	1	0	1	1	1 1142	1 1142	0 2
13	1	0	0	1	2 1096	1 4654	0 2078
14	1	0	0	0	0 484	1 327	0 2282
15	1	0	0	1	0 7138	1 8414	0 22
16	1	0	1	1	0 301	3 6754	0 276
17	1	0	1	0	0 4148	3 5646	0 2
18	1	0	1	1	0 2968	3 311	0 2
19	1	1	1	1	1 914	1 914	3 2314
20	1	1	1	0	1 8804	1 8804	3 1368
21	1	1	1	1	1 9024	1 9024	3 1626
22	1	1	0	1	2 1326	1 8496	3 2648
23	1	1	0	0	2 2344	2 0026	3 1654
24	1	1	0	1	2 043	1 967	3 1016
25	1	1	1	1	2 1446	1 706	3 2122
26	1	1	1	0	2 2512	2 0006	3 1616
27	1	1	1	1	2 0866	1 9156	3 1134
28	0	1	1	1	2 7508	2 7508	3 6546
29	0	1	1	0	2 3946	2 3946	1 9196
30	0	1	1	1	2 0174	2 0174	1 9342
31	0	1	0	1	3 0406	1 0612	0 5092
32	0	1	0	0	2 7644	1 27	0 6386
33	0	1	0	1	1 4216	0 9338	1 92
34	0	1	1	1	2 9926	1 685	1 65
35	0	1	1	0	2 1928	2 2308	0 6822
36	0	1	1	1	0 5776	2 8234	1 5372
37	0	0	1	1	2 5098	2 5098	0 8464
38	0	0	1	0	1 7186	1 7186	0 7998
39	0	0	1	1	2 1068	2 1068	0 2194
40	0	0	0	1	2 68	0 758	0 2994
41	0	0	0	0	2 6418	0 797	0 2194
42	0	0	0	1	2 7184	0 533	0 23
43	0	0	1	1	0 9134	2 1766	0 4888
44	0	0	1	0	0 6368	3 1654	0 2648
45	0	0	1	1	0 3682	3 47	0 2
46	0	1	1	1	1 1658	1 1658	0 5362
47	0	1	1	0	0 4618	0 4616	1 4096
48	0	1	1	1	0 7398	0 7398	2 0484
49	0	1	0	1	0 82	0 6866	0 2334
50	0	1	0	0	0 28	1 7106	0 36
51	0	1	0	1	0 8774	1 1824	1 5094
52	0	1	1	1	0 24	1 228	0 8748
53	0	1	1	0	0 7644	1 7008	0 8134
54	0	1	1	1	0 8588	1 5402	1 0536
55	1	1	1	1	3 1308	3 1308	3 4174
56	1	1	1	0	2 519	2 519	2 8906
57	1	1	1	1	2 0958	2 0958	2 2884
58	1	1	0	1	2 9888	2 9572	1 664
59	1	1	0	0	2 2232	1 934	1 3342
60	1	1	0	1	2 4144	1 0632	1 202
61	1	1	1	1	2 9048	2 463	1 6598
62	1	1	1	0	1 8972	2 2908	1 3996
63	1	1	1	1	0 5404	2 1684	2 628
64	1	0	1	1	3 4296	3 4296	0 5704
65	1	0	1	0	2 429	2 429	0 8828
66	1	0	1	1	2 4018	2 4018	0 633
67	1	0	0	1	3 4824	1 1052	0 2596
68	1	0	0	0	2 964	0 8522	0 4166
69	1	0	0	1	2 914	0 746	0 2
70	1	0	1	1	2 8552	2 1066	0 3596
71	1	0	1	0	2 2648	2 9762	0 2298
72	1	0	1	1	0 4956	3 3344	0 227

Table 1 Experiment Results for Parameter Setting of the APTCR Rule (Continued)

Instance	μ	α	γ	ρ	k1	k2	k3
73	1	1	1	1	2 4696	2 4696	0 4692
74	1	1	1	0	1 7862	1 7862	0 2802
75	1	1	1	1	1 3384	1 3384	1 0708
76	1	1	0	1	1 8682	0 7224	0 2318
77	1	1	0	0	2 2778	1 9988	0 2
78	1	1	0	1	0 56	1 62	0 2
79	1	1	1	1	0 3184	3 7618	0 2464
80	1	1	1	0	0 4892	2 573	0 2134
81	1	1	1	1	0 3444	3 8996	0 2

Table 2 Experiment Results for Parameter Setting of the SA

No	ITemp	Alpha	StepMax	TMax	Avg. Error from Optimal
1	1	1	1	1	0 3039
2	1	1	1	0	0 2270
3	1	1	1	1	0 1765
4	1	1	0	1	0 3099
5	1	1	0	0	0 2299
6	1	1	0	1	0 1721
7	1	1	1	1	0 2992
8	1	1	1	0	0 2387
9	1	1	1	1	0 1596
10	1	0	1	1	0 3132
11	1	0	1	0	0 2330
12	1	0	1	1	0 1690
13	1	0	0	1	0 3114
14	1	0	0	0	0 2320
15	1	0	0	1	0 1662
16	1	0	1	1	0 3141
17	1	0	1	0	0 2462
18	1	0	1	1	0 1736
19	1	1	1	1	0 3237
20	1	1	1	0	0 2345
21	1	1	1	1	0 1643
22	1	1	0	1	0 2907
23	1	1	0	0	0 2407
24	1	1	0	1	0 1570
25	1	1	1	1	0 3309
26	1	1	1	0	0 2364
27	1	1	1	1	0 1770
28	0	1	1	1	0 3067
29	0	1	1	0	0 2420
30	0	1	1	1	0 1657
31	0	1	0	1	0 3066
32	0	1	0	0	0 2283
33	0	1	0	1	0 1480
34	0	1	1	1	0 3306
35	0	1	1	0	0 2388
36	0	1	1	1	0 1769
37	0	0	1	1	0 3151
38	0	0	1	0	0 2453
39	0	0	1	1	0 1632
40	0	0	0	1	0 3162
41	0	0	0	0	0 2419
42	0	0	0	1	0 1732
43	0	0	1	1	0 3267
44	0	0	1	0	0 2386
45	0	0	1	1	0 1684
46	0	1	1	1	0 3179
47	0	1	1	0	0 2350
48	0	1	1	1	0 1612
49	0	1	0	1	0 3221
50	0	1	0	0	0 2246
51	0	1	0	1	0 1726
52	0	1	1	1	0 3304
53	0	1	1	0	0 2256
54	0	1	1	1	0 1728
55	1	1	1	1	0 3044
56	1	1	1	0	0 2339
57	1	1	1	1	0 1740
58	1	1	0	1	0 3380
59	1	1	0	0	0 2337

Table 2 Experiment Results for Parameter Setting of the SA (Continued)

No	TTemp	Alpha	StepMax	TMax	Avg. Error from Optimal
60	1	1	0	1	0.1704
61	1	1	1	1	0.3225
62	1	1	1	0	0.2276
63	1	1	1	1	0.1773
64	1	0	1	1	0.3233
65	1	0	1	0	0.2359
66	1	0	1	1	0.1620
67	1	0	0	1	0.3199
68	1	0	0	0	0.2503
69	1	0	0	1	0.1790
70	1	0	1	1	0.3282
71	1	0	1	0	0.2388
72	1	0	1	1	0.1617
73	1	1	1	1	0.3382
74	1	1	1	0	0.2382
75	1	1	1	1	0.1625
76	1	1	0	1	0.3160
77	1	1	0	0	0.2230
78	1	1	0	1	0.1805
79	1	1	1	1	0.3056
80	1	1	1	0	0.2338
81	1	1	1	1	0.1775

Table 3 Experiment Results for Parameter Setting of the MetaRaPS (p%=0.1)

No	priority	restrict	improve	priority*restrict	priority*improve	restrict*improve	Avg. Performance
1	1	1	1	1	1	1	0.07
2	1	1	0.5	1	0.5	0.5	0.05
3	1	1	0	1	0	0	0.05
4	1	1	0.5	1	0.5	0.5	0.05
5	1	1	1	1	1	1	0.05
6	1	0.75	1	0.75	1	0.75	0.02
7	1	0.75	0.5	0.75	0.5	0.375	0.02
8	1	0.75	0	0.75	0	0	0.02
9	1	0.75	0.5	0.75	0.5	0.375	0.02
10	1	0.75	1	0.75	1	0.75	0.02
11	1	0.5	1	0.5	1	0.5	0.03
12	1	0.5	0.5	0.5	0.5	0.25	0.04
13	1	0.5	0	0.5	0	0	0.03
14	1	0.5	0.5	0.5	0.5	0.25	0.04
15	1	0.5	1	0.5	1	0.5	0.04
16	1	0.25	1	0.25	1	0.25	0.03
17	1	0.25	0.5	0.25	0.5	0.125	0.03
18	1	0.25	0	0.25	0	0	0.03
19	1	0.25	0.5	0.25	0.5	0.125	0.03
20	1	0.25	1	0.25	1	0.25	0.03
21	1	0	1	0	1	0	0.03
22	1	0	0.5	0	0.5	0	0.03
23	1	0	0	0	0	0	0.03
24	1	0	0.5	0	0.5	0	0.03
25	1	0	1	0	1	0	0.03
26	1	0.25	1	0.25	1	0.25	0.08
27	1	0.25	0.5	0.25	0.5	0.125	0.08
28	1	0.25	0	0.25	0	0	0.08
29	1	0.25	0.5	0.25	0.5	0.125	0.08
30	1	0.25	1	0.25	1	0.25	0.08
31	1	0.5	1	0.5	1	0.5	0.09
32	1	0.5	0.5	0.5	0.5	0.25	0.09
33	1	0.5	0	0.5	0	0	0.09
34	1	0.5	0.5	0.5	0.5	0.25	0.09
35	1	0.5	1	0.5	1	0.5	0.09
36	1	0.75	1	0.75	1	0.75	0.33
37	1	0.75	0.5	0.75	0.5	0.375	0.33
38	1	0.75	0	0.75	0	0	0.32
39	1	0.75	0.5	0.75	0.5	0.375	0.33
40	1	0.75	1	0.75	1	0.75	0.33
41	1	1	1	1	1	1	0.39
42	1	1	0.5	1	0.5	0.5	0.39
43	1	1	0	1	0	0	0.39
44	1	1	0.5	1	0.5	0.5	0.40
45	1	1	1	1	1	1	0.39

APPENDIX C : RESULTS FOR SMALL SIZE PROBLEMS FOR ARSP

Table 1 Comparison Results for APTCR and SA (One replication)

Instance	Mu	Alfa	Gama	Ro	k1	k2	k3	$Error_{APTCR}$	Time (sec.)	$Error_{SA}$	Max	Min	Time (sec.)
1	3	1.5	0	0.1	2.68	2.47	2.16	0.18	0.001	0.00	0.00	0.00	0.060
2	3	1.5	0	0.3	2.30	2.16	1.88	0.55	0	0.06	0.09	0.00	0.062
3	3	1.5	2	0.1	2.93	1.42	1.04	0.23	0.001	0.21	0.33	0.12	0.090
4	3	1.5	2	0.3	2.59	1.42	0.84	0.04	0.006	0.18	0.32	0.07	0.091
5	3	2	0	0.1	2.12	2.47	0.92	0.14	0.005	0.00	0.00	0.00	0.068
6	3	2	0	0.3	1.91	2.16	0.92	1.33	0.004	0.67	0.67	0.67	0.081
7	3	2	2	0.1	2.43	1.42	0.35	0.21	0.006	0.22	0.35	0.15	0.092
8	3	2	2	0.3	2.25	1.42	0.35	2.29	0.006	3.46	4.99	0.37	0.092
9	6	1.5	0	0.1	2.86	2.03	2.82	0.00	0.002	0.00	0.00	0.00	0.055
10	6	1.5	0	0.3	2.36	1.73	2.49	0.00	0.003	0.07	0.12	0.00	0.057
11	6	1.5	2	0.1	2.98	0.98	1.50	0.22	0.004	0.03	0.06	0.00	0.063
12	6	1.5	2	0.3	2.52	0.98	1.27	0.27	0.004	0.14	0.19	0.03	0.063
13	6	2	0	0.1	2.34	2.03	0.65	0.03	0.002	0.00	0.00	0.00	0.056
14	6	2	0	0.3	1.99	1.73	0.65	0.91	0.004	0.23	0.27	0.00	0.057
15	6	2	2	0.1	2.49	0.98	0.19	0.13	0.005	0.46	1.02	0.15	0.071
16	6	2	2	0.3	2.17	0.98	0.19	0.39	0.006	0.25	0.44	0.14	0.077

Table 2 Comparison Results for SA and MetaRaPS(One replication)

Instance	Mu	Alfa	Gama	Ro	SA	Max	Min	Time(sec.)	MR	Max	Min	Time(sec.)
1	3	1.5	0	0.1	0.01	0.03	0.00	0.118	0.03	0.03	0.03	0.097
2	3	1.5	0	0.3	0.17	0.27	0.00	1.296	0.00	0.00	0.00	1.378
3	3	1.5	2	0.1	0.17	0.21	0.14	1.489	0.03	0.09	0.00	1.576
4	3	1.5	2	0.3	0.06	0.06	0.03	1.962	0.01	0.03	0.00	2.100
5	3	2	0	0.1	0.00	0.00	0.00	1.927	0.06	0.20	0.00	2.024
6	3	2	0	0.3	0.02	0.17	0.00	1.585	0.00	0.00	0.00	1.711
7	3	2	2	0.1	1.52	1.64	0.56	1.641	0.05	0.16	0.00	1.737
8	3	2	2	0.3	0.37	0.37	0.37	1.998	0.29	0.35	0.27	2.101
9	6	1.5	0	0.1	0.00	0.00	0.00	1.952	0.00	0.00	0.00	2.065
10	6	1.5	0	0.3	0.03	0.06	0.00	0.972	0.00	0.00	0.00	1.044
11	6	1.5	2	0.1	0.06	0.07	0.01	1.174	0.01	0.02	0.00	1.244
12	6	1.5	2	0.3	0.08	0.15	0.00	1.514	0.02	0.04	0.00	1.597
13	6	2	0	0.1	0.00	0.00	0.00	1.536	0.00	0.00	0.00	1.635
14	6	2	0	0.3	0.09	0.13	0.00	1.159	0.00	0.00	0.00	1.232
15	6	2	2	0.1	0.03	0.07	0.00	1.271	0.02	0.04	0.01	1.369
16	6	2	2	0.3	0.04	0.04	0.04	1.435	0.00	0.01	0.00	1.517

APPENDIX D : RESULTS FOR LARGE SIZE PROBLEMS FOR ARSP

Table 1 Comparison Results for APTCR and SA (One replication)

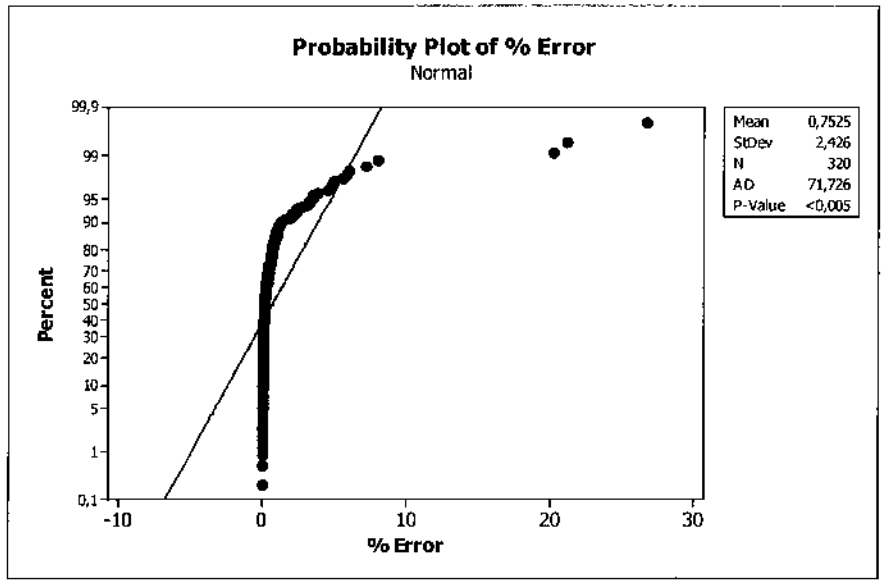
Instance	Mu	Alfa	Gama	Ro	k1	k2	k3	TWT_{APTCR}	Time (sec.)	TWT_{SA}	Max	Min	Time (sec.)	Avg (SA APTCR)/APTCR
1	3	1.5	0	0.1	2.68	2.47	2.16	10560	0.006	18312	19080	17640	0.499	0.73
2	3	1.5	0	0.3	2.30	2.16	1.88	9360	0.038	15732	16320	14520	0.505	0.68
3	3	1.5	2	0.1	2.93	1.42	1.04	2290	0.006	4036	4234	3725	0.984	0.76
4	3	1.5	2	0.3	2.59	1.42	0.84	1686	0.006	3980	4194	3749	0.985	1.36
5	3	2	0	0.1	2.12	2.47	0.92	4680	0.005	9744	10440	8520	0.582	1.08
6	3	2	0	0.3	1.91	2.16	0.92	3000	0.004	7872	8280	7320	0.615	1.62
7	3	2	2	0.1	2.43	1.42	0.35	507	0.005	2129	2381	1872	0.987	3.20
8	3	2	2	0.3	2.25	1.42	0.35	342	0.005	1845	1954	1614	0.985	4.39
9	6	1.5	0	0.1	2.86	2.03	2.82	17400	0.002	23460	24880	23160	0.416	0.35
10	6	1.5	0	0.3	2.36	1.73	2.49	13680	0.002	22488	23160	21600	0.441	0.64
11	6	1.5	2	0.1	2.98	0.98	1.50	8519	0.004	13144	13684	12718	0.621	0.54
12	6	1.5	2	0.3	2.52	0.98	1.27	10299	0.004	13986	14452	13690	0.635	0.36
13	6	2	0	0.1	2.34	2.03	0.65	12240	0.003	18816	19560	18120	0.446	0.54
14	6	2	0	0.3	1.99	1.73	0.65	6840	0.003	15276	15720	14520	0.473	1.23
15	6	2	2	0.1	2.49	0.98	0.19	6437	0.004	10842	11206	10296	0.681	0.68
16	6	2	2	0.3	2.17	0.98	0.19	3547	0.004	7667	8021	7073	0.722	1.16

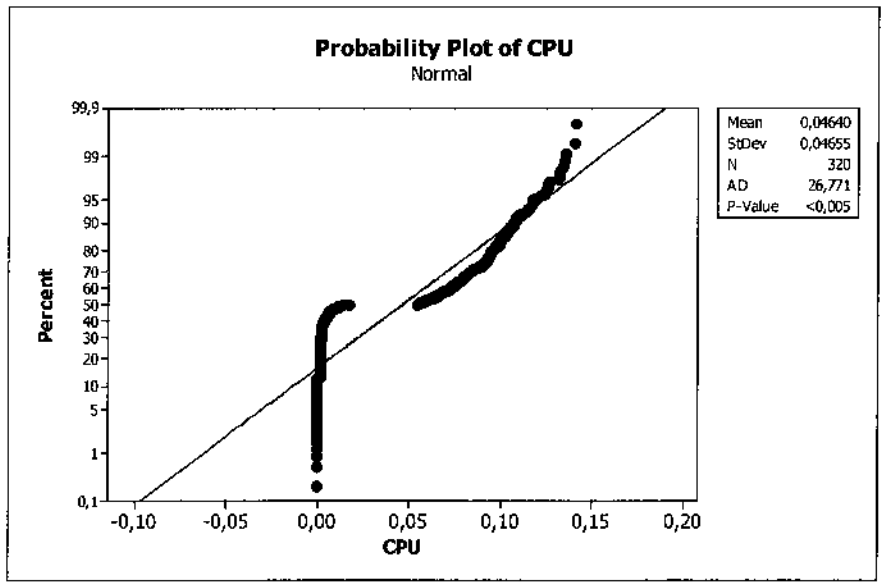
Table 2 Comparison Results for SA and MetaRaPS (One replication)

Instance	Avg Relative SA APTCR	Max	Min	Time(sec.)	Max	Min	Avg Relative MetaRaPS	Max	Min	Time(sec.)	Max	Min
1	0.03	0.18	0.10	0.96	0.98	0.95	0.00	0.00	0.00	1.32	1.34	1.31
2	0.01	0.00	0.15	1.05	1.07	1.03	0.00	0.06	0.00	1.56	1.61	1.54
3	0.00	0.00	0.00	1.72	1.75	1.70	0.04	0.07	0.01	1.99	2.01	1.97
4	0.00	0.11	0.04	1.69	1.71	1.67	0.02	0.00	0.00	2.04	2.08	2.02
5	0.00	0.00	0.00	1.20	1.22	1.18	0.15	0.25	0.05	1.75	1.78	1.73
6	0.28	0.23	0.33	1.24	1.25	1.23	0.00	0.00	0.00	1.87	1.88	1.85
7	0.00	0.00	0.00	1.78	1.80	1.75	0.12	0.18	0.03	2.07	2.09	2.04
8	0.40	0.30	0.61	1.69	1.72	1.67	0.00	0.00	0.00	2.12	2.14	2.10
9	0.00	0.00	0.00	0.86	0.88	0.84	0.11	0.09	0.09	0.82	0.84	0.80
10	0.00	0.00	0.01	0.89	0.92	0.88	0.01	0.06	0.00	1.11	1.13	1.09
11	0.04	0.28	0.00	1.22	1.24	1.21	0.00	0.00	0.00	1.57	1.60	1.55
12	0.06	0.14	0.07	1.24	1.25	1.23	0.00	0.00	0.00	1.74	1.76	1.71
13	0.00	0.00	0.00	0.92	0.94	0.90	0.15	0.19	0.12	1.12	1.15	1.10
14	0.00	0.52	0.00	0.96	1.02	0.94	0.02	0.00	0.00	1.35	1.38	1.33
15	0.05	0.47	0.03	1.34	1.38	1.32	0.00	0.00	0.00	1.65	1.68	1.63
16	0.13	0.21	0.14	1.43	1.45	1.41	0.00	0.00	0.00	1.98	2.02	1.96

APPENDIX E : STATISTICAL TEST RESULTS FOR SCHEDULING ALGORITHMS

Figure 1 Normality Tests for APTCR -SA Comparison
Small Size Problems(n=12)





Tests for APTCR -SA Comparison in Small Size Problems(n=12)

Kruskal-Wallis Test: % Error versus Method

```
Kruskal-Wallis Test on % Error

Method     N   Median  Ave Rank      Z
DDW-R    160  0.20266     174.4   2 68
SA       160  0.09735     146.6  -2.68
Overall  320              160.5

H = 7.18  DF = 1  P = 0.007
H = 7.21  DF = 1  P = 0.007  (adjusted for ties)
```

Kruskal-Wallis Test: CPU versus Method

```
Kruskal-Wallis Test on CPU

Method     N    Median  Ave Rank       Z
DDW-R    160  0.001000      80.5  -15 47
SA       160  0.091750     240.5   15.47
Overall  320              160.5

H = 239.25  DF = 1  P = 0.000
H = 241.19  DF = 1  P = 0.000  (adjusted for ties)
```

Test for Equal Variances: % Error versus Method

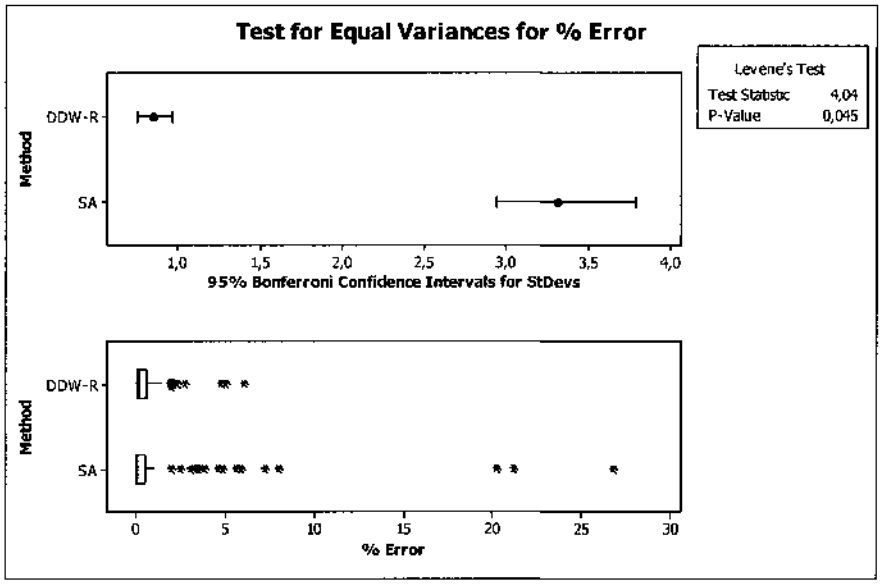
```
95% Bonferroni confidence intervals for standard deviations

Method    N    Lower    StDev    Upper
 DDW-R  160  0.75518  0.85040  0.97188
    SA  160  2.93976  3.31041  3.78328


F-Test (Normal Distribution)
Test statistic = 0.07; p-value = 0 000


Levene's Test (Any Continuous Distribution)
Test statistic = 4.04; p-value = 0.045
```

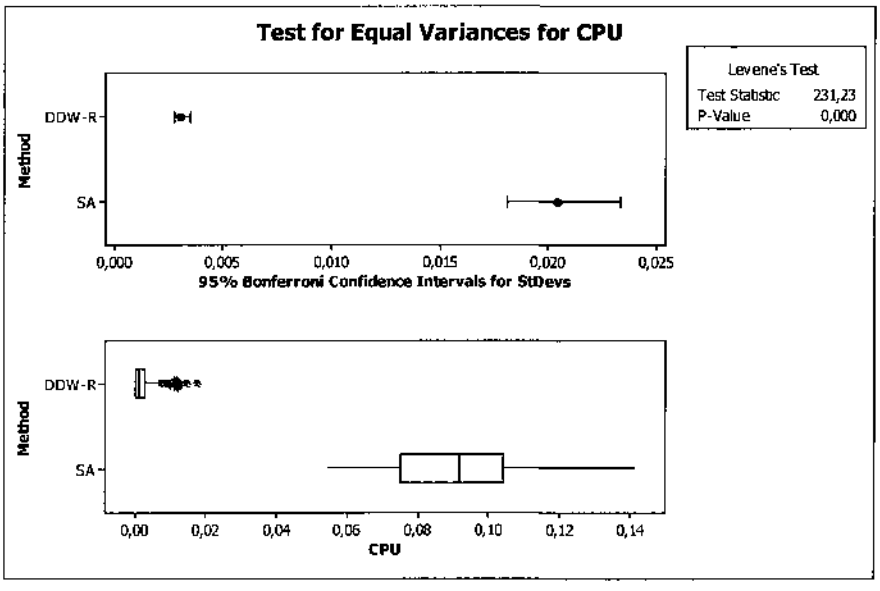
Test for Equal Variances: CPU versus Method

```
95% Bonferroni confidence intervals for standard deviations

Method    N      Lower      StDev      Upper
 DDW-R  160  0.0026969  0 0030370  0.0034708
    SA  160  0.0181035  0.0203861  0.0232981


F-Test (Normal Distribution)
Test statistic = 0.02; p-value = 0.000


Levene's Test (Any Continuous Distribution)
```

```
Test statistic = 231.23; p-value = 0.000
```

Figure 2 Equal Variance Tests for APTCR-SA Comparison in Small Size Problems





APTCR Test Results

Kruskal-Wallis Test: % Error versus M

```
Kruskal-Wallis Test on % Error

M          N  Median  Ave Rank      Z
3         80  0.3881      93.7   3.60
6         80  0.1250      67.3  -3.60
Overall  160              80.5

H = 12.95  DF = 1  P = 0.000
H = 12.96  DF = 1  P = 0.000  (adjusted for ties)
```

Kruskal-Wallis Test: % Error versus A

```
Kruskal-Wallis Test on % Error

A          N  Median  Ave Rank      Z
1.5       80  0.1270      61.6  -5.15
2.0       80  0.4416      99.4   5.15
Overall  160              80.5

H = 26.54  DF = 1  P = 0.000
H = 26.55  DF = 1  P = 0.000  (adjusted for ties)
```

Kruskal-Wallis Test: % Error versus G

```
Kruskal-Wallis Test on % Error

G          N  Median  Ave Rank      Z
0         80  0.1484      74.7  -1.59
2         80  0.2219      86.3   1.59
Overall  160              80.5

H = 2.54  DF = 1  P = 0.111
H = 2.54  DF = 1  P = 0.111  (adjusted for ties)
```

Kruskal-Wallis Test: % Error versus R

```
Kruskal-Wallis Test on % Error

R          N  Median  Ave Rank      Z
0.1       80  0.1228      68.0  -3.41
0.3       80  0.3542      93.0   3.41
Overall  160              80.5

H = 11.60  DF = 1  P = 0.001
H = 11.61  DF = 1  P = 0.001  (adjusted for ties)
```

SA Test Results

Kruskal-Wallis Test: % Error versus M

```
Kruskal-Wallis Test on % Error

M          N   Median  Ave Rank      Z
3         80  0.37500      97.4   4.60
6         80  0.05323      63.6  -4.60
Overall  160               80.5

H = 21.18  DF = 1  P = 0.000
H = 21.40  DF = 1  P = 0.000  (adjusted for ties)
```

Kruskal-Wallis Test: % Error versus A

```
Kruskal-Wallis Test on % Error

A          N   Median  Ave Rank      Z
1.5       80  0.05484      69.8  -2.92
2.0       80  0.19050      91.2   2.92
Overall  160               80.5

H = 8.55  DF = 1  P = 0.003
H = 8.64  DF = 1  P = 0.003  (adjusted for ties)
```

Kruskal-Wallis Test: % Error versus G

```
Kruskal-Wallis Test on % Error

G          N   Median  Ave Rank      Z
0         80  0.03077      53.9  -7.26
2         80  0.38678     107.1   7.26
Overall  160               80.5

H = 52.64  DF = 1  P = 0.000
H = 53.20  DF = 1  P = 0.000  (adjusted for ties)
```

Kruskal-Wallis Test: % Error versus R

```
Kruskal-Wallis Test on % Error

R          N   Median  Ave Rank      Z
0.1       80  0.05183      70.3  -2.79
0.3       80  0.16417      90.7   2.79
Overall  160               80.5

H = 7.78  DF = 1  P = 0.005
H = 7.87  DF = 1  P = 0.005  (adjusted for ties)
```

Tests for APTCR -SA Comparison in Large Size Problems(n=60)

Kruskal-Wallis Test: Relative TWT versus Method

```
Kruskal-Wallis Test on Relative TWT

Method      N  Median  Ave Rank       Z
DDW-R     160  0.5273      80.5  -15.47
SA        160  1.0000     240.5   15.47
Overall   320             160.5

H = 239.25  DF = 1  P = 0.000
H = 273.43  DF = 1  P = 0.000  (adjusted for ties)
```

Kruskal-Wallis Test: Relative TWT versus M

```
Kruskal-Wallis Test on Relative TWT

M           N  Median  Ave Rank      Z
3          80  1.4715     112.3   8.69
6          80  0.5478      48.7  -8.69
Overall   160              80.5

H = 75.43  DF = 1  P = 0.000
```

Kruskal-Wallis Test: Relative TWT versus A

```
Kruskal-Wallis Test on Relative TWT

A           N  Median  Ave Rank      Z
1.5        80  0.6213      58.3  -6.06
2.0        80  1 2920     102 7   6.06
Overall   160              80.5

H = 36.69  DF = 1  P = 0.000
```

Kruskal-Wallis Test: Relative TWT versus G

```
Kruskal-Wallis Test on Relative TWT

G           N  Median  Ave Rank      Z
0          80  0.7851      72.3  -2.25
2          80  1.0045      88.7   2.25
Overall   160              80.5

H = 5.06  DF = 1  P = 0.025
```

Kruskal-Wallis Test: Relative TWT versus R

```
Kruskal-Wallis Test on Relative TWT

R          N  Median  Ave Rank      Z
0.1       80  0.6009      66.0  -3.97
0.3       80  1.0861      95.0   3.97
Overall  160              80.5

H = 15.72  DF = 1  P = 0.000
```

Kruskal-Wallis Test: CPU versus Method

```
Kruskal-Wallis Test on CPU

Method     N    Median  Ave Rank       Z
APTCR    160  0.005000      80.5  -15.47
SA       160  0.608000     240.5   15.47
Overall  320              160.5

H = 239.25  DF = 1  P = 0.000
H = 240.77  DF = 1  P = 0.000  (adjusted for ties)
```

Tests for SA-MetaRaPS Comparison in Small Size Problems(n=12)

Kruskal-Wallis Test: Relative Error versus Method

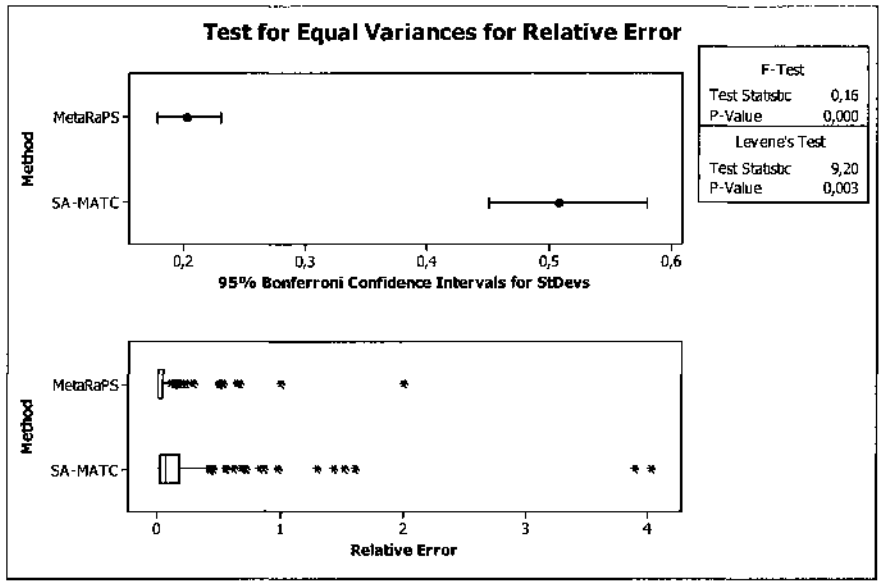
```
Kruskal-Wallis Test on Relative Error

Method      N    Median  Ave Rank      Z
MetaRaPS  160  0.008370     129.0  -6.10
SA-APTCR  160  0.061534     192.0   6.10
Overall   320              160.5

H = 37.17  DF = 1  P = 0.000
H = 38.37  DF = 1  P = 0.000  (adjusted for ties)
```

Test for Equal Variances: Relative Error versus Method

```
95% Bonferroni confidence intervals for standard deviations

  Method    N     Lower     StDev     Upper
MetaRaPS  160  0.179393  0.202011  0.230867
 SA-APTCR  160  0.450057  0.506801  0.579195


F-Test (Normal Distribution)
Test statistic = 0.16; p-value = 0.000


Levene's Test (Any Continuous Distribution)
```

```
Test statistic = 9.20; p-value = 0.003
```

Figure 3 Equal Variance Tests for SA-MetaRaPS Comparison in Small Size Problems



Tests for SA-MetaRaPS Comparison in Large Size Problems(n= 60)

Kruskal-Wallis Test: Relative Difference versus Method

```
Kruskal-Wallis Test on Relative Difference

Method       N       Median  Ave Rank      Z
MetaRaPS   160  0.000000000     143.0  -3.38
SA-APTCR   160  0.016623239      178.0   3.38
Overall    320                  160.5

H = 11.41  DF = 1  P = 0.001
H = 13.04  DF = 1  P = 0.000  (adjusted for ties)
```

APPENDIX F: TEST RESULTS FOR RESCHEDULING ALGORITHMS

Test Results for Job Arrival Disruption Repair Algorithms

Kruskal-Wallis Test: Error versus Arrival for MetaRE

```
Kruskal-Wallis Test on Error

Arrival    N   Median  Ave Rank      Z
Early     30  0.01803      23.7  -3.03
Late      30  2.95075      37.3   3.03
Overall   60               30.5

H = 9.19  DF = 1  P = 0.002
H = 9.36  DF = 1  P = 0.002  (adjusted for ties)
```

Kruskal-Wallis Test: Error versus Weight for MetaRE

```
Kruskal-Wallis Test on Error

Weight     N  Median  Ave Rank      Z
0.2       10  0.7605      32.1   0.32
0.5       10  0.3487      26.6  -0.78
0.8       10  0.5988      28.4  -0.41
0.92      10  0.5498      29.9  -0.13
0.95      10  0.7510      31.4   0.19
0.98      10  0.8760      34.6   0.81
Overall   60               30.5

H = 1.33  DF = 5  P = 0.932
H = 1.35  DF = 5  P = 0.929  (adjusted for ties)
```

Kruskal-Wallis Test: Error versus Arrival for BestINSERT

```
Kruskal-Wallis Test on Error_1

Arrival    N  Median  Ave Rank      Z
Early     30  0.3324      30.4  -0.07
Late      30  0.1605      30.6   0.07
Overall   60               30.5

H = 0.00  DF = 1  P = 0.947
H = 0.00  DF = 1  P = 0.946  (adjusted for ties)
```

Kruskal-Wallis Test: Error versus Weight for BestINSERT

```
Kruskal-Wallis Test on Error_1

Weight     N       Median  Ave Rank      Z
0.2       10  0.000000000      15.0  -3.07
0.5       10  0.000316656      21.4  -1.81
0.8       10  0.149241281      29.6  -0.18
0.92      10  0.534511580      32.0   0.30
0.95      10  0.785661723      38.5   1.60
0.98      10  1.737360140      46.5   3.17
Overall   60                   30.5

H = 21.24  DF = 5  P = 0.001
H = 22.34  DF = 5  P = 0.000  (adjusted for ties)
```

Kruskal-Wallis Test: Error versus Arrival for SEPRE

```
Kruskal-Wallis Test on Error_2

Arrival    N  Median  Ave Rank      Z
Early     30   2.633      29.9  -0.27
Late      30   5.891      31.1   0.27
Overall   60              30.5

H = 0.07  DF = 1  P = 0.790
H = 0.07  DF = 1  P = 0.790  (adjusted for ties)
```

Kruskal-Wallis Test: Error versus Weight for SEPRE

```
Kruskal-Wallis Test on Error_2

Weight     N  Median  Ave Rank      Z
0.2       10   4.219      32.1   0.32
0.5       10   3.806      29.5  -0.20
0.8       10   3.184      29.9  -0.12
0.92      10   3.184      29.9  -0.12
0.95      10   3.806      29.5  -0.20
0.98      10   4.219      32.1   0.32
Overall   60              30.5

H = 0.26  DF = 5  P = 0.998
H = 0.26  DF = 5  P = 0.998  (adjusted for ties)
```

Kruskal-Wallis Test: Error of MetaRE and BestINSERT for low weights

```
Algorithm     N       Median  Ave Rank      Z
BestINSERT   20  0.000000000      14.9  -3.02
MetaRE       20  0.721721818      26.1   3.02
Overall      40                   20.5

H = 9.10  DF = 1  P = 0.003
H = 10.01  DF = 1  P = 0.002  (adjusted for ties)
```

Test Results for Priority Disruption Repair Algorithms

Kruskal-Wallis Test: Relative Error versus Weight for SHUFFLE

```
Kruskal-Wallis Test on Relative Error

Weight     N  Median  Ave Rank      Z
0.2       10  0.6922      12.8  -1.21
0.5       10  0.8314      12.2  -1.47
0.8       10  2.5304      21.6   2.68
Overall   30              15.5

H = 7.23  DF = 2  P = 0.027
H = 7.24  DF = 2  P = 0.027  (adjusted for ties)
```

Kruskal-Wallis Test: Relative Error versus Urgency for SHUFFLE

```
Kruskal-Wallis Test on Relative Error

Urgency    N  Median  Ave Rank      Z
High      15  0.6126      12.7  -1.76
Low       15  1.5218      18.3   1.76
Overall   30              15.5

H = 3.11  DF = 1  P = 0.078
H = 3.11  DF = 1  P = 0.078  (adjusted for ties)
```

Kruskal-Wallis Test: Relative Error_1 versus Weight for BestINSERT

```
Kruskal-Wallis Test on Relative Error_1

Weight     N  Median  Ave Rank      Z
0.2       10  0.5540      14.8  -0.31
0.5       10  0.1432      11.6  -1.72
0.8       10  1.9329      20.1   2.02
Overall   30              15.5

H = 4.76  DF = 2  P = 0.093
H = 4.80  DF = 2  P = 0.091  (adjusted for ties)
```

Kruskal-Wallis Test: Relative Error_1 versus Urgency for BestINSERT

```
Kruskal-Wallis Test on Relative Error_1

Urgency    N  Median  Ave Rank      Z
High      15  0.5540      15.0  -0.33
Low       15  0.5540      16.0   0.33
Overall   30              15.5

H = 0.11  DF = 1  P = 0.740
H = 0.11  DF = 1  P = 0.739  (adjusted for ties)
```

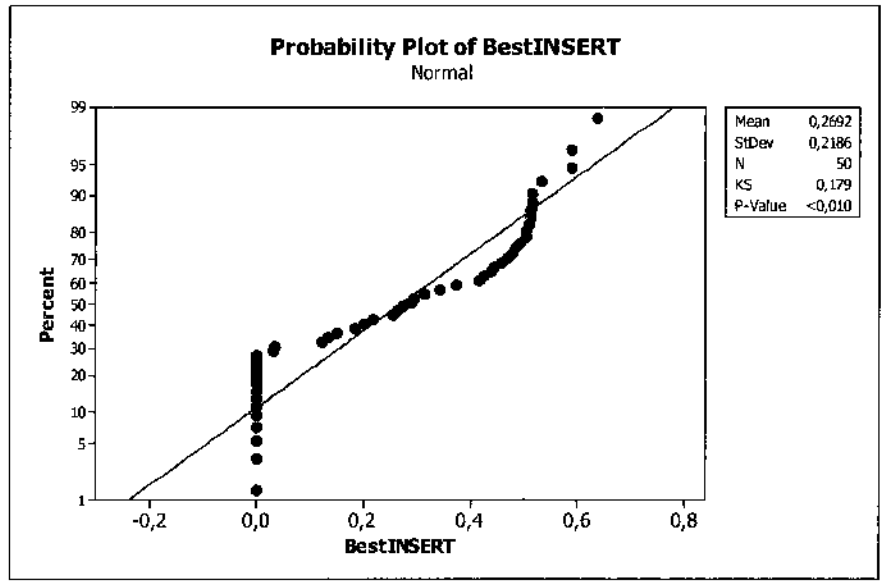
Kruskal-Wallis Test: Relative Error_2 versus Weight for MetaRE

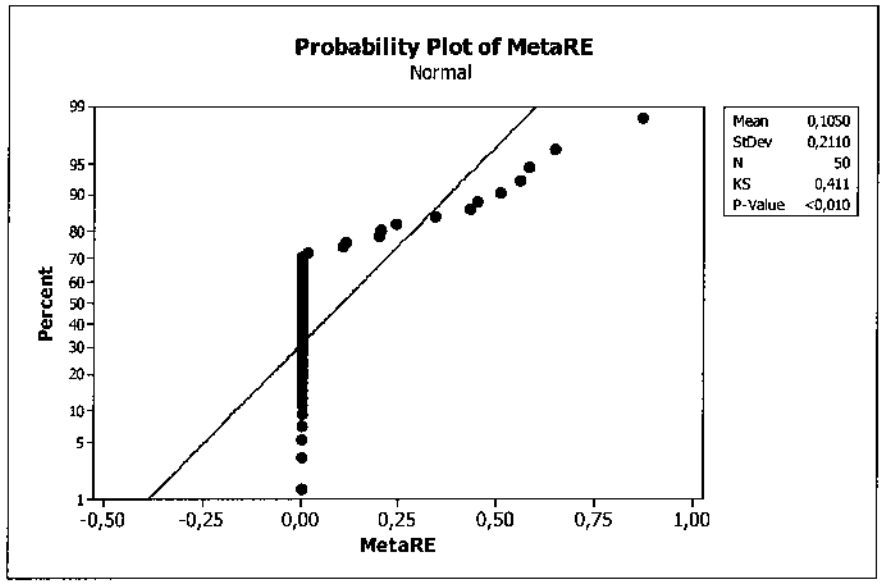
```
Kruskal-Wallis Test on Relative Error_2

Weight    N  Median  Ave Rank      Z
0.2      10  1.4101      17.8   1.01
0.5      10  0.2063      14.9  -0.26
0.8      10  0.1311      13.8  -0.75
Overall  30              15.5

H = 1.10  DF = 2  P = 0.576
H = 1.10  DF = 2  P = 0.576  (adjusted for ties)
```

Kruskal-Wallis Test: Relative Error_2 versus Urgency for MetaRE

```
Kruskal-Wallis Test on Relative Error_2

Urgency   N  Median  Ave Rank      Z
High     15  0.5562      15.2  -0.19
Low      15  0.2096      15.8   0.19
Overall  30              15.5

H = 0.03  DF = 1  P = 0.852
H = 0.03  DF = 1  P = 0.852  (adjusted for ties)
```

Figure 4 Normality Tests for Priority Disruption Repair Algorithms (n=60)





Test Results for Priority Disruption Repair Algorithms (n=60)

Kruskal-Wallis Test: BestINSERT versus Weight

Weight	N	Median	Ave Rank	Z
0.1	10	0.486524908	38.5	3.15
0.3	10	0.463406959	36.2	2.60
0.5	10	0.287605001	30.1	1.12
0.7	10	0.032470626	15.2	-2.50
0.9	10	0.000000000	7.5	-4.37
Overall	50		25.5	

H = 34.58 DF = 4 P = 0.000
H = 35.35 DF = 4 P = 0.000 (adjusted for ties)

Kruskal-Wallis Test: BestINSERT versus Disruption Time

Disruption Time	N	Median	Ave Rank	Z
Early	25	0.1990	24.3	-0.60
Late	25	0.2944	26.7	0.60
Overall	50		25.5	

H = 0.36 DF = 1 P = 0.548
H = 0.37 DF = 1 P = 0.543 (adjusted for ties)

Kruskal-Wallis Test: MetaRE versus Weight

Weight	N	Median	Ave Rank	Z
0.1	10	0.000000000	18.5	-1.70
0.3	10	0.000000000	18.5	-1.70
0.5	10	0.000000000	18.5	-1.70
0.7	10	0.000000000	26.6	0.27
0.9	10	0.478103929	45.4	4.83
Overall	50		25.5	

H = 25.61 DF = 4 P = 0.000
H = 40.85 DF = 4 P = 0.000 (adjusted for ties)

Kruskal-Wallis Test: MetaRE versus Disruption Time

Disruption Time	N	Median	Ave Rank	Z
Early	25	0.000000000	27.6	1.00
Late	25	0.000000000	23.4	-1.00
Overall	50		25.5	

H = 1.00 DF = 1 P = 0.318
H = 1.59 DF = 1 P = 0.207 (adjusted for ties

VITA

Sezgin KAPLAN

Engineering Management and System Engineering
Old Dominion University, Norfolk /VA
e-Mail: skaplan@odu.edu, sezginkaplan64@yahoo.com

RESEARCH INTEREST :

Planning and Scheduling, Applied Optimization, Multi Criteria Decision Making, Logistics and Supply Chain Optimization.

EDUCATION :

M.S. : 2005 – 2007 Department of Industrial Engineering
Gazi University
Ankara, Turkey
M.A. : 2000 – 2004 Department of Economy
Hacettepe University
Ankara, Turkey
B.S. : 1996 – 2000 Department of Industrial Engineering
Bilkent University
Ankara, Turkey

THESIS :

Kaplan S. (2007). *Evaluation of the Equipment Investment Projects In Air Defense Sector by Fuzzy AHP*. Gazi University, Applied Science Institution, Industrial Engineering M.S. Program, Ankara, TURKEY.
Kaplan S. (2004). *Market Structure and Innovation.* Hacettepe University, Social Sciences Institution, Economy M.A. Program, Ankara, TURKEY.

WORK EXPERIENCE :

Oct 2002 – Jun 2005 Production Planning and Control Officer
TUAF 3rd Air Supply and Maintenance Center Command, Ankara, TURKEY
Jun 2005 – Jun 2008 Work Load Planner Officer
TUAF 3rd Air Supply and Maintenance Center Command, Ankara, TURKEY