8-10-2010

# Minimizing Makespan for Hybrid Flowshops with Batch, Discrete Processing Machines and Arbitrary Job Sizes

Yanming Zheng
*Florida International University*, yzhen001@fiu.edu

**DOI:** 10.25148/etd.FI11050312

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

# MINIMIZING MAKESPAN FOR HYBRID FLOWSHOPS WITH BATCH, DISCRETE PROCESSING MACHINES AND ARBITRARY JOB SIZES

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSHOPY

in

INDUSTRIAL AND SYSTEMS ENGINEERING

by

Yanming Zheng

2011

To: Dean Amir Mirmiran
College of Engineering and Computing

This dissertation, written by Yanming Zheng, and entitled Minimizing Makespan for Hybrid Flowshops with Batch, Discrete Processing Machines and Arbitrary Job Sizes, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

_____
Shih-Ming Lee

_____
Ronald Giachetti

_____
Purushothaman Damodaran

_____
Chin-Sheng Chen, Co-Major Professor

_____
Syed M. Ahmed, Co-Major Professor

Date of Defense: August 10, 2010

The dissertation of Yanming Zheng is approved.

_____
Dean Amir Mirmiran
College of Engineering and Computing

_____
Interim Dean Kevin O'Shea
University Graduate School

Florida International University, 2011

ABSTRACT OF THE DISSERTATION

MINIMIZING MAKESPAN FOR HYBRID FLOWSHOPS WITH BATCH, DISCRETE PROCESSING MACHINES AND ARBITRARY JOB SIZES

by

Yanming Zheng

Florida International University, 2011

Miami, Florida

Professor Chin-Sheng Chen, Co-Major Professor

Professor Syed M. Ahmed, Co-Major Professor

This research aims at a study of the hybrid flow shop problem which has parallel batch-processing machines in one stage and discrete-processing machines in other stages to process jobs of arbitrary sizes. The objective is to minimize the makespan for a set of jobs. The problem is denoted as: $FF|batch1, s_j|Cmax$.

The problem is formulated as a mixed-integer linear program. The commercial solver, AMPL/CPLEX, is used to solve problem instances to their optimality. Experimental results show that AMPL/CPLEX requires considerable time to find the optimal solution for even a small size problem, i.e., a 6-job instance requires 2 hours in average.

A bottleneck-first-decomposition heuristic (BFD) is proposed in this study to overcome the computational (time) problem encountered while using the commercial solver. The proposed BFD heuristic is inspired by the shifting bottleneck heuristic. It decomposes the entire problem into three sub-problems, and schedules the sub-problems one by one. The proposed BFD heuristic consists of four major steps: formulating sub-

problems, prioritizing sub-problems, solving sub-problems and re-scheduling. For solving the sub-problems, two heuristic algorithms are proposed; one for scheduling a hybrid flow shop with discrete processing machines, and the other for scheduling parallel batching machines (single stage). Both consider job arrival and delivery times. An experiment design is conducted to evaluate the effectiveness of the proposed BFD, which is further evaluated against a set of common heuristics including a randomized greedy heuristic and five dispatching rules. The results show that the proposed BFD heuristic outperforms all these algorithms.

To evaluate the quality of the heuristic solution, a procedure is developed to calculate a lower bound of makespan for the problem under study. The lower bound obtained is tighter than other bounds developed for related problems in literature.

A meta-search approach based on the Genetic Algorithm concept is developed to evaluate the significance of further improving the solution obtained from the proposed BFD heuristic. The experiment indicates that it reduces the makespan by 1.93 % in average within a negligible time when problem size is less than 50 jobs.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

# 1 INTRODUCTION

## 1.1 Background and Motivation

As industries are facing more and more intense competition, many manufacturing companies adapt newer systems, such as hybrid flow shops (HFS) which combine the flow shop with parallel machines. Batch processing machines are also used to simultaneously process multiple jobs to improve efficiency. The hybrid flow shop with batch processing machines in parallel is denoted as HFPB by Amin-Naseri (2009). The problem under study considers a variant case of HFPB, denoted as $FF_m|batch1,s|Cmax$, which represents a hybrid flow shop with parallel identical batch processing machines in exactly one stage and discrete processing machines in other stages to process jobs with arbitrary size.

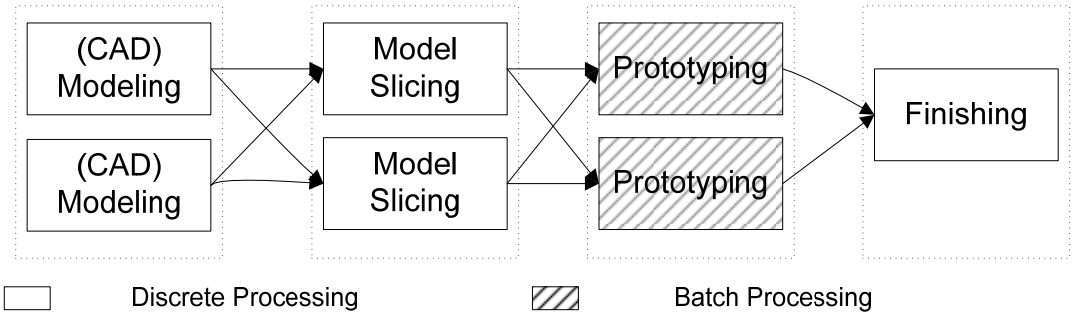



Figure 1-1 A Rapid Prototyping Process

One of the typical applications of this problem is in rapid prototyping. The rapid prototyping refers to a class of technologies that can automatically construct physical

models from Computer-Aided Design (CAD) data. These "three dimensional printers" allow designers to quickly create tangible prototypes of their designs, rather than just two-dimensional pictures. Prototyping plays an essential role in the development of appliances, machines, cars and many other items, small or large we encounter in our daily life. It can reduce development costs, shorten lead time and test configuration. Although several rapid prototyping techniques exist, all employ the same basic four-step process which is shown in Figure 1-1. The steps include:

1. Model Simulation: The design to be prototyped is transformed from a one-dimensional image to a three-dimensional likeness. This is usually done using computer-aided design (CAD) software such as AutoCAD, and converted into the STL format which is considered a standard in the rapid prototyping process.

2. Model Slicing: A pre-processing software is run to slice the STL model into a number of layers, depending on the “build” technique.

3. Prototyping: A rapid prototyping machine is used to build the prototype one layer at a time from polymers, paper, or powdered metal. Usually a prototyping machine can prototype multiple items as a batch simultaneously. The time required is determined by the longest time required by all the items in the batch.

4. Finishing: The final step is post-processing. This involves removing the prototype from the machine and detaching any supports.

The scheduling of rapid prototyping is to make two important and intertwined decisions: (1) how to group jobs to form batches for prototyping, and (3) how to sequence the jobs and batches so that the makespan of processing all prototypes will be minimized. Since different prototype has different design, which results in different size and requires

different processing times, the number of prototypes to be built simultaneously in a prototyping machine cannot be predetermined, and hence increase the complexity of batching and scheduling.

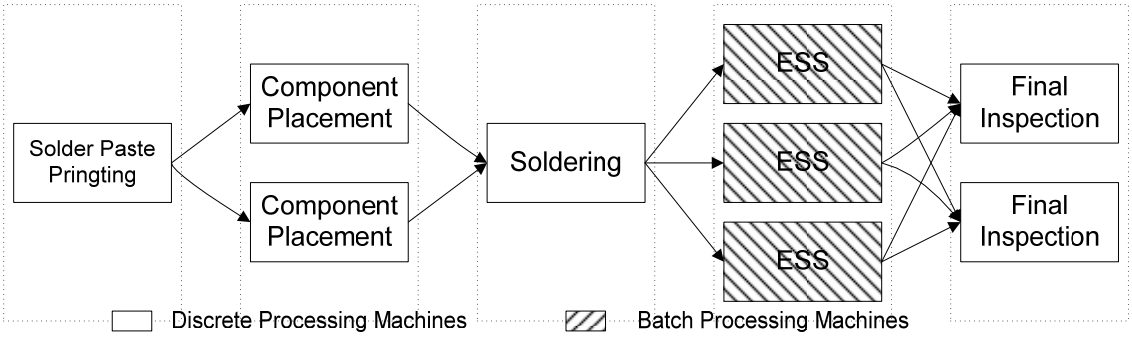



Figure 1-2 PCB assembly process

Another example is the printed circuit board (PCB) assembly (Figure 1-2). The solder paste is first applied on a bare PCB. The PCB is later populated with appropriate components by a component placement machine. Later, the assembly is soldered in a convection oven. After soldering, PCBs are placed in an environmental stress screening chamber (ESS) for an environmental test, and then sent to the next operation for final inspection. PCB manufacturers, especially contract electronics manufacturers, usually have more than one machine at some stages of processing. In most of the processing, each machine can only process one PCB at a time, while in environmental stress testing one ESS chamber can test several PCBs simultaneously. The number of PCBs that can be processed at a time depends on the chamber's capacity as well as the size of the PCBs.

Applications of the scheduling problem under study can also be found in various industries, for example chemical industry and steel industry.

## 1.2 Problem Statement, Research Objectives and Deliverables

### 1.2.1 Problem Statement

The problem under study can be described as follows: We are given multiple stages of machines. Each stage has one or more identical machines in parallel. One stage consists of parallel batch processing machines, while all other stages consist of discrete processing machines. The batching stage can be any stage along the flow line. For all jobs to be processed, job processing times and job sizes are arbitrary. Each job should go through all stages and should be processed by only one machine in each stage. The process route is identical for all jobs. In the batching stage, machines are able to process two or more jobs simultaneously as long as the total size of jobs in a batch does not exceed the machine capacity. All jobs in the same batch start and finish at the same time. The batch processing time is determined by the longest job processing time of all jobs in the batch. The objective is to find a set of batches, the schedule of jobs and batches in each stage, so that the completion time of the last job is minimized.

To study this scheduling problem, three issues are raised:

1. The batching strategy

   Minimizing the number of batches or maximizing the size of each batch is usually the batching strategy for scheduling single and parallel batch processing machines. However, minimizing the number of batches may yield a good solution only when all the jobs are available at the same time. When jobs arrive at different time, a batch might close even though there is still space available, so as to avoid delaying the processing of the batch. The

question to answer is when to close the batch, or when to stop waiting for next job to join. On the other hand, the dispatching rule used to prioritize jobs should be determined, too.

2. The impact of shop configuration on batching strategy

   As mentioned above, one of the scheduling decisions to make is how to form the batches. It is easy to understand that different batching strategies might yield different scheduling results. However, does batching strategy vary with shop configuration? Will the dispatching rule change when the batching stage position changes from the first stage to the last stage? If yes, what dispatching rules should be considered for different shop configurations?

3. The interaction between discrete hybrid flow shop and parallel batching machines

   The problem under study is an integration of the discrete HFS scheduling problem and the parallel batching machines scheduling problem. To get a good solution, we cannot simply split the problem into several sub-problems and get solution for each sub-problem, since job completion times in one stage will affect the scheduling in the next stage. Hence, integrated solutions should be explored to consider the interaction between the hybrid flow shop and the parallel batching machines.

This research aims to study above issues, and therefore fill the research gap in this field.

### 1.2.2 The Complexity Analysis

If the number of stages is 3, the number of machines in each stage is 1, and all job sizes and capacity for all machines are equal to 1, the problem is then reduced to F3||Cmax, with time complexity function known to be NP-hard. If there is only one stage, and all job sizes and machine capacities are equal to 1, the problem is reduced to Pm||Cmax which is also NP-hard. It can be inferred that the problem under study is at least NP-hard, which means that finding an optimum solution in a reasonable time is unlikely, and heuristic methods are highly recommended.

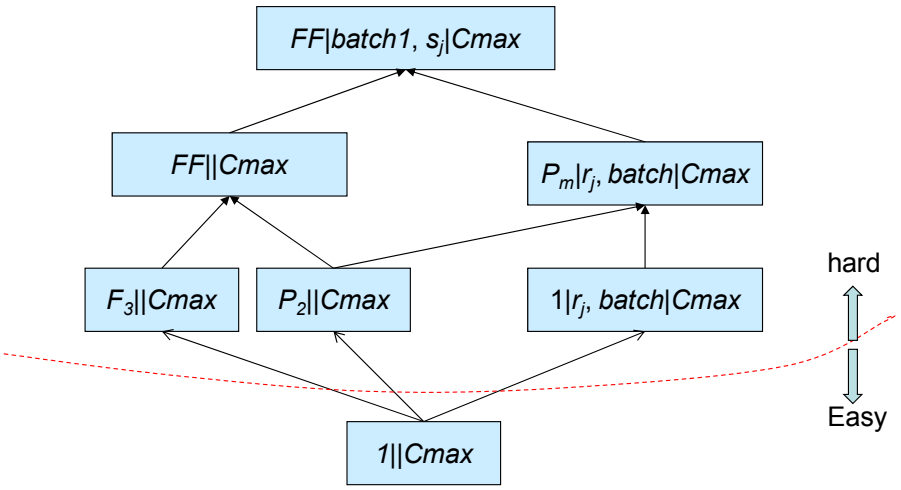



Figure 1-3 Complexity Analysis

### 1.2.3 Research Objectives and Deliverables

The objective of this study is to mathematically formulate the problem under study and develop a solution approach to effectively solve the problem of practical size. Since the problem under study is NP-hard, to solve such problems, the run-time of an

exact algorithm, for example enumeration schemes or branch and bound, increases exponentially with the instance size, implying that usually only small instances can be solved in practice. In contrast with the exact methods, a heuristic would require much less time to find an approximate solution. However, the solution quality might be compromised. In this dissertation, the effectiveness of the heuristic approach is evaluated by measuring the run time and comparing its solution with the lower bound. Figure 1-4 presents the different approaches developed for this research.

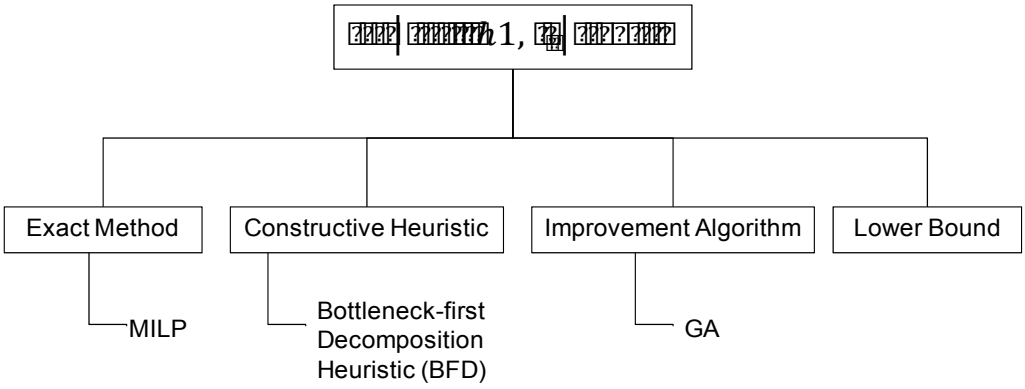



Figure 1-4 Overview of the Methodology

In this research a mixed-integer linear program is first formulated and solved using a commercial solver to obtain optimum solution. However, as the problem is NP–hard, optimum solution can be obtained in reasonable time for only small-size problem instances. A constructive heuristic approach is proposed to solve the problem under study approximately. The performance of the approach is measured by the computational cost and the solution quality. A meta-heuristic, genetic algorithm (GA), is also proposed to explore the significance of improvement on the solution produced by the constructive

heuristic. A procedure to calculate the lower bound of the makespan is developed to assess the quality of the solution obtained from various solution approaches proposed. Since optimal solution is difficult to obtain for even small-size problems, having a good lower bound is important for comparing the effectiveness of different solution approaches. In this research, the derived lower bound is used as the benchmark to evaluate the solution quality of heuristics whenever an optimal solution is not available.

The deliverables of this study include: (1) A mixed-integer model, (2) a lower-bound procedure, (3) a heuristic approach, and (4) a meta-heuristic approach.

## 1.3 Research Assumptions

The main assumptions are: (1) machines and jobs are available from the beginning, (2) neither job splitting nor preemption is allowed, (3) The buffers between stages are unlimited, (4) job processing times and sizes are deterministic and known in advance, (5) machine breakdowns are not considered, (6) there is no waiting time limit between any two stages, (7) machine set up time is not considered, (8) all machines within the same stage are identical, (9) the batch processing time equals the longest processing time among all jobs in the batch, (10) all jobs in a batch start and finish at the same time, and (11) the capacity and job size is assumed one dimension.

## 1.4 Significance and Contribution

The significance of this research can be addressed in two perspectives. First, the problem under study has a growing popularity of applications in different fields of industry, for example rapid prototyping, PCB assembly, metal working, etc. Actually, any production line related to a hybrid flow shop with batch and discrete processing

machines might encounter this scheduling problem. A solution approach to the problem under study will benefit the scheduling in these fields, and improve productivity of production line. Second, this research differs from most previous work done on HFS scheduling in that it considers a hybrid flow shop with both discrete and batch processing machines and allow jobs to have arbitrary sizes. Similar but simpler problems have never been solved by any exact algorithm within reasonable computational time and for a number of jobs that is of practical interest.

The research contribution can be summarized as: (1) a category of scheduling problem, $FF|batch1, s_j|Cmax$, is mathematically modeled, (2) a tight lower bound is derived for this problem, and (3) a solution approach to solving this problem of practical size effectively and efficiently is proposed.

## 1.5 Dissertation Organization

The rest of the dissertation is organized as follows. Chapter 2 presents a literature review on existing research. Chapter 3 introduces a mixed-integer model. A lower bound is given in Chapter 4. In Chapter 5, a heuristic approach built on the decomposition method and the shifting bottleneck procedure is presented and its performance is compared to that of a set of common heuristics and dispatching rules. Chapter 6 describes a genetic algorithm to further improve the solutions. Conclusions are drawn in Chapter 7.

# 2 LITERATURE REVIEW

## 2.1 Introduction

The problem under study can be considered as the integration of two basic scheduling problems: (1) the classical Hybrid Flow Shop problem denoted as $FF_m \mid\mid C_{max}$, and (2) Parallel batch processing machines problem, denoted as $P_m \mid batch \mid C_{max}$. The literature review is organized as below. Section 2.2 focuses on the HFS scheduling. Section 2.3 reviews parallel batching machines scheduling problems. Section 2.4 addresses the HFS with both discrete and batch processing machines. Section 2.5 covers the decomposition methods. Section 2.6 discusses the genetic algorithm. A summary is given in Section 2.7.

## 2.2 Hybrid Flow Shop Scheduling

### 2.2.1 Hybrid Flow Shop with Identical Job Ready Times

The scheduling problem to minimize the makespan on a hybrid flow shop is known to be NP-hard in the strong sense (Hoogeveen 1996). Various solution approaches have been proposed and can be grouped into three categories: exact methods, heuristics and meta-heuristics (Table 2-1).

Arthanari and Ramamurthy (1971) define the HFS problem, and first suggest a branch and bound method to this problem. After that, the branch and bound method has been explored by many researchers to get optimal solutions for small size instances. Among all these, Neron et al. (2001) describe an exact approach which outperforms all

previous ones and reports the optimal solution of small-sized instances with up to 15 jobs, 5 stages, and 3 machines in each stage.

Table 2-1 Previous Research on Hybrid Flow Shop with Identical Ready Times

| Classes of Solution Approaches | | Previous Research | |
|---|---|---|---|
| | | Two Stages | Three Stages and Above |
| Exact methods | | Arthanari and Ramamurthy(1971)<br>Salvador (1973)<br>Gupta et al. (1997)<br>Portmann et al. (1998)<br>Haouari et al. (2006) | Brah and Hunsucker (1991)<br>Rajendran and Chaudhuri (1992)<br>Perregaard (1995)<br>Portmann et al. (1998)<br>Carlier and Néron (2000)<br>Moursli and Pochet (2000)<br>Neron et al. (2001) |
| Heuristics | Bottleneck-based heuristics | | Adler et al. (1993)<br>Yang (1998)<br>Chen (1998)<br>Acero-Domínguez and Paternina-Arboleda (2004)<br>Lee et al. (2004)<br>Paternina-Arboleda et al. (2008)<br>Chen et al. (2009) |
| | Other heuristics | Narasimhan and Panwalker (1984)<br>Narasimhan and Mangiameli (1987)<br>Gupta and Tunc (1991) (1994)<br>Gupta et al. (1997) | Wittrock (1985)<br>Wittrock (1988)<br>Ding and Kittichartphayak (1994)<br>Hunsucker and Shah (1994)<br>Santos et al. (1996)<br>Guinet and Solomon (1996)<br>Brah and Loo (1999)<br>Gupta et al. (2002)<br>Caricato and Grieco (2007) |
| Meta-heuristics | | | Nowicki and Smutnicki (1998)<br>Gourgand et al. (1999)<br>Engin and Doyen (2004)<br>Jin et al. (2002)<br>Vishwanathan et al. (2007)<br>Oulamara et al. (2009) |

Since the exact methods, i.e. branch and bound, is computationally expensive and has been proved impractical for even modestly sized problems, it usually appeals to apply heuristic algorithms for large-scale HFS problems, to achieve near-optimal solutions. Therefore, starting from 1980's, many studies have been conducted on solving two-stage HFS problems with heuristics. Narasimhan and Panwalker (1984) solve a variant of the HFS problem that had different machines at the second stage by using the Cumulative Minimum Deviation rule. Narasimhan and Mangiameli (1987) propose a Generalized Cumulative Minimum Deviation rule to solve HFS problems. Gupta et al. (1991; 1994; 1997) research two-stage HFS problems with parallel machines at one of the stages and have extended their research to the two-stage HFS problems with parallel machines at both stages, and separable setup and removal times.

For the HFS problem with more than two stages, the well known early researches are conducted by Wittrock (1985; 1988). The author develops a periodic heuristic algorithm for minimizing the makespan by focusing on job loading and time allocating. He also presents a more flexible non-periodic heuristic algorithm for the same problem by taking three steps: machine allocation, job sequencing and timing. Ding and Kittichartphayak (1994) develops three heuristics for HFS problems. The heuristics are extensions of Campbell-Dudek-Smith (CDS) by Campbell (1970) and a heuristic by Gupta (1972). Hunsucker and Shah (1994) evaluate six priority rules used in an HFS problem with a constrained total number of jobs in the system. The objectives are makespan, mean flow time and maximum flow time. They conclude that the SPT dispatching rule is superior for makespan and mean flow time. However, a similar study done by Guinet and Solomon (1996) lead to a different conclusion. The authors declare

that NEH heuristic of Nawaz, Enscore, and Ham with Largest Processing Time First rule is the best for Cmax instead of SPT. Santos et al. (1996) evaluate four heuristics algorithms used in HFS and concluded that 'a non-delay-type scheduling procedure which utilizes a "good" starting permutation of job orderings at first stage of an HFS should produce a makespan which is not appreciably worse than that produced by an optimal general schedule. Brah and Loo (1999) expand five better-performing flow shop heuristics, and use regressions to investigate heuristic performance and the effects of problem characteristics. Gupta et al.(2002) study multiple-stage HFS problems with parallel machines at each stage with multiple criteria. It is assumed that processing times are controllable and due dates are assignable. The authors generalize well-known approaches for the heuristic solution of classical problems and propose heuristic algorithms based on job insertion techniques and iterative algorithms on local search. Caricato and Grieco (2007) extends the traditional HFS scheduling problem to the case in which jobs are due to follow strict precedence constraints and batch assignment constraints and the parallel machines at a stage are served by a bottleneck machine. A variant of the well-known Travelling salesman problem (TSP) is used to develop an efficient heuristic solution for the problem. Furthermore, a simple insertion heuristic based on the TSP model of the problem is tested.

A category of heuristics called bottleneck-based heuristics become popular in the past two decades in scheduling HFS problem. The main idea behind bottleneck-based heuristic is to find the bottleneck stage and to optimize the whole system performance based on exploiting the bottleneck stage. It usually comprises three steps: (i) bottleneck identification, (ii) scheduling of jobs at the bottleneck stage, and (iii) scheduling of jobs

at non-bottleneck stages. Adler et al. (1993) present a bottleneck heuristic algorithm to for scheduling a HFS with non-identical machines in paper company with due date related objective. Yang (1998) proposes a bottleneck-based heuristic to minimize the total weighted tardiness for the HFS scheduling problem. Chen (1998) suggests a bottleneck-based group scheduling procedure to solve flow line cell scheduling problems. Lee et al. (2004) develop a bottleneck-focused algorithm to minimize total tardiness of a HFS scheduling problem. In their algorithm, the ready times are iteratively updated using information of the schedule obtained in the previous iterations. Acero-Domínguez and Paternina-Arboleda (2004), Paternina-Arboleda et al. (2008) develop heuristics based on Theory of Constraint (TOC) by Goldratt (1992) and minimize the makespan of a hybrid flow shop by exploiting the bottleneck stage. They claim that this heuristic is with smaller variance and requires less computational effort than other bottleneck based algorithms. Chen et al. (2009) develop a bottleneck-based heuristic (BBFFL) to solve a flexible flow line with unrelated parallel machines , with the objective of minimizing the makespan, the total tardiness and number of tardy jobs.

Different meta-heuristic approaches are proposed to improve the solutions, for example Tabu search by Nowicki and Smutnicki (1998), simulated annealing algorithms by Gourgand et al. (1999), Jin et al. (2002) and Vishwanathan et al. (2007), genetic algorithms by Jin et al. (2002) and Oulamara et al. (2009) , and artificial immune system by Orhan Engin (2004).

### 2.2.2 Hybrid Flow Shop with Non-Identical Job Ready Times

Although there have been many studies on hybrid flow shop scheduling problems, most of such studies deal with problems in which ready times of jobs are not considered, that is, ready times of jobs are identical or equal to zero. Researches on HFS problems with non-identical job ready times are very limited. Cheng et al. (2001) propose a shifting bottleneck heuristic to minimize the $L_{max}$ on the HFS with non-identical job ready times. The heuristic decomposes the HFS into $m$ parallel-machine scheduling problems to be solved one by one. Each parallel-machine problem is approximately solved by applying a property of its reversibility in the proposed heuristic. The authors declare that their heuristic produces optimal or quite near optional solutions within short computational time. Gupta et al. (2002) propose heuristics for the HFS with non-identical job ready times, controllable processing times and assignable due dates. The authors also use a list scheduling method to deal with non-identical job ready times. Tang and Liu (2009) derive a lower bound and develop dynamic programming-based heuristic algorithms to minimize the makespan on a two-machine flow shop with batching and release time.

## 2.3 Parallel Batching Machines Scheduling

### 2.3.1 Parallel Batching Machines with Identical Job Ready Times

Batch processing by parallel machine is proven to be NP-hard in strong sense (Lee 1992). Research efforts in this area can be classified using the following 3 independent criteria: (1) constant or varying batch-processing times, (2) presence or absence of job sizes as a variable, and (3) presence or absence of incompatible job families. Under the first criterion, if the batch processing times are not constant, the

processing time of a batch depends on the jobs that constitute a batch. Otherwise, the batch processing times are independent of these jobs. Under the second criterion, if job sizes are not considered, it is assumed that all the jobs have the same size and therefore the machine capacity is given by the maximum number of jobs it can process simultaneously. On the other hand, if job sizes are not identical, the machine capacity is given by the maximum number of size units the machine can process simultaneously. Under the third criterion, if incompatible job families are present, the jobs assigned to the same batch must belong to the same family, and usually jobs belonging to the same family share a common processing time, which determine the batch processing time. If incompatible job families are not considered, the batch processing time is given by the maximum processing time of the jobs assigned to the batch. The problem under study in this dissertation assumes that batch processing times are varying and determined by the composition of the batches, the job sizes are non-identical, and there is no incompatible job present.

Lee et al.(1992) observe that there exists an optimal schedule in which all jobs are pre-assigned into batches according to the BLPT rule: rank the jobs in non-increasing order of processing times, and then batch the jobs by successively placing as many as possible jobs with the largest processing times into the same batch. Remy (2004) presents an algorithm which approximates the makespan for parallel batching machines with ratio 2. Lin (2004) studies the parallel batch scheduling problem to minimize the maximum lateness and the number of tardy jobs. First dynamic programming algorithms are designed for finding optimal solutions to the two problems, and then several heuristics are developed to solve the same problems. Chang et al. (2004) develop a simulated

annealing algorithm to minimize the makespan for capacitated parallel batch processing machines that process jobs with arbitrary sizes. Kashan et al (2008) propose a hybrid genetic heuristic (HGH) to solve the same problem. The author claims that the HGH outperforms the simulated annealing (SA) approach by Chang et al. (2004). Shao et al. (2008) propose a solution approach based on neural networks on the same problem. Xu (2007) proposes a genetic algorithm based on random keys (RKGA) to minimize the makespan on parallel non-identical batch processing machines. Damodaran et al. (2007) develop several heuristics to minimize the makespan on identical parallel batching machines. Damodaran (2008) also develops a GA algorithm for the same problem.

### 2.3.2 Parallel Batching Machines with Non-identical Job Ready Times

Chung et al. (2008) propose a mathematical model and three heuristics to minimize the makespan on parallel batching machines with non-identical job arrival times and job sizes. The authors propose a sequential approach in which batches are formed first and scheduled second. To form the batches, they propose the Modified Delay algorithm which incorporates the merits of the DELAY heuristic solution procedure proposed by Lee (1999). Damodaran et al. (2008) and Vélez-Gallego (2009) propose several heuristics and meta-heuristics for the same problem. Two heuristics, JS1 and JS2, are proposed to form batches. The authors claim that their scheduling rule with JS1 and JS2 heuristics outperform the other heuristics, including the Modified Delay heuristic addressed by Chung et al. (2008).

Other than minimizing the makespan, there is also research effort made on due date relate objectives. Chiang (2008) proposes a local search-based heuristic to minimize

maximum lateness on identical parallel batching machines scheduling problem considering incompatible job families and dynamic job arrivals. Li (2004) present a polynomial time approximation scheme (PTAS ) for minimizing maximum lateness on identical parallel batch processing machines scheduling problem with ready times and delivery times.

## 2.4 Flow Shop with Both Discrete and Batch Processing Machines

In this environment, jobs are routed through a flow shop which has batch processing machines in some stages, and discrete processing machines in the other stages. Table 2-2 shows major research papers in this area grouped into different categories.

Table 2-2 Previous Research on Flow shop With Discrete and Batching Machines

| | Two stages | Multiple stage |
|---|---|---|
| Flow shop | Tang (2009)<br>Sung (2002)<br>Su (2003) | AHMADI (1992) |
| Hybrid flow shop | Kim et al. (2009)<br>Bellanger (2009)<br>Bellanger (2008) | Vishwanathan (2007)<br>Amin-Naseri (2009) |

AHMADI (1992) minimizes the makespan (Cmax) and the sum of completion times (ΣCj) in two-machine flow shop scenario with one processor being a batching machine and the other being a discrete machine. The batch processing time is assumed to be independent of the composition of the batch. The authors proposes Full Batch-LPT rules for the flow shop scenario with batching machine followed by discrete machine, and

SPT-Full Batch rules for the reversed machine environment with discrete machine followed by batching machine. Both heuristics are claimed to yield optimal solution. Sung (2002) considers a two-machine flow shop where a discrete processing machine is followed by a batch processing machine and a finite number of jobs arrive dynamically at the first machine. In Sung's heuristic, the dynamic shortest processing time (DSPT) rule is applied to schedule the first machine, and then FOE(n, c) heuristic is adopted to form batches and schedule in the second machine. A tight worst-case error bound is also derived. Su (2003) analyzes the problem of minimizing Cmax of a two-stage flow shop having limited waiting time constraints, with a batch processor in stage 1 and a single processor in stage 2. A heuristic algorithm and a mixed integer program are proposed. Vishwanathan (2007) presents a hybrid flow shop scheduling problem with an interior stage consists of a batch processing operation and all other stages being discrete-parts processing. Different classic flow shop heuristics (CDS, NEH, Palmer's heuristic) are applied at the first stage, coupled with a FIFO progression throughout the shop. The experiment result shows that Palmer and CDS outperform NEH on most problem instances. Bellanger (2009) considers a two-stage hybrid flow shop problem in which the first stage contains several identical discrete machines, and the second stage contains several identical batching machines which can process several compatible tasks simultaneously in a batch. A heuristics family denoted as H-FCBLPT along with their worst cases analysis is developed to minimize the makespan. Tang et al. (2009) derive a lower bound and develop dynamic programming-based heuristic algorithms to solve the scheduling problem in a two-machine flow shop environment with batching and release time, whose objective is to minimize the makespan. They apply DSPT to schedule the

first stage of single discrete machine and then use dynamic programming to find an optimal job batching for the second stage based on given job sequence from stage 1. Kim (2009) studies the scheduling problem of a two-stage hybrid flow shop with identical parallel machines at the first stage and a single batch processing machine at the second stage, subject to non-identical ready time and product-mix ratio constraint. Three heuristics, forward heuristic, backward heuristic and iterative heuristic, are presented to minimize the makespan. Bellanger (2008) presents a exact method based on Branch & Bound for a two-stage hybrid flow shop with parallel discrete machines in the first stage and parallel batching machines in the second stage, with the objective to minimize the makespan. All jobs of the same batch have to be compatible. Recently, Amin-Naseri (2009) develops three heuristic algorithm to minimize makespan in a hybrid flow shop environment with parallel uniform batching machines in some stages and parallel identical discrete machines in other stages. It is assumed that the job sizes are identical. Three heuristics, H1, H2 and H3, which are inspired by Johnson's rule, CDS and theory of constraints (TOC) accordingly, are developed to minimize the makespan. A lower bound and a three dimensional genetic algorithm (3DGA) is developed for evaluating the performance of the proposed heuristics. Kim et al. (2009) study the scheduling problem of a two-stage hybrid flow shop with identical parallel machines at the first stage and a single batch processing machine at the second stage, subject to non-identical ready time and product-mix ratio constraint.

## 2.5 Decomposition Approaches

### 2.5.1 Introduction of Decomposition Approaches

Decomposition approaches have been used widely in the operations research fields as a manner of modeling and analyzing complex systems. The idea behind it is that the resolution of certain critical (bottleneck) sub-problems will enable us to construct the solution to the remaining parts of the original problem relatively easily. Therefore if we successfully formulate and solve the bottleneck sub problems, it will be able to obtain a near-optimal solution relative easily since the solutions to the remaining sub-problems are determined to a great extent by the solutions to the bottleneck sub problems. If sub problems differ in criticality, the most critical or most constraining sub problems will be solved first to ensure the quality of the solution. Based on this idea, decomposition methods attempt to develop solutions to complex problems by decomposing them into a number of smaller sub-problems which are more tractable and easier to understand. Solutions are developed for each sub-problem individually, and then integrated to form a solution to the original problem. The solutions obtained may be exact or approximate, depending on the nature of the procedure used and the problem under study. Ovacik (1997) classified decomposition methods into three major types:

1) Temporal decomposition schemes that decompose problems based on time,
2) Entity decomposition schemes that decompose problems based on work centers or machines and,
3) Hybrid decomposition schemes, which are combination of temporal decomposition schemes and entity decomposition schemes.

They discuss a variety of issues regarding the development of successful decomposition procedures: how to decompose the shop problem into appropriate sub-problems, how to model the interactions between sub-problems, how to choose an effective solution procedure for the sub-problems, and how to design an appropriate control strategy that determines in what order the sub problems should be solved and how intermediate solutions should be revised to improve solution quality. Most of the researches focus on minimizing Lmax on a job shop.

### 2.5.2 Shifting Bottleneck Procedures

One of the most successful decomposition algorithms applied to shop scheduling is the shifting bottleneck procedure originally developed by Adams (1988), which is designed for the classical *Jm* | | *Cmax* problem. The procedure decomposes the job shop problem into single machine sub-problems. In each iteration it aims at fixing the schedule for the current bottleneck machine. It achieves this goal by scheduling the set of unscheduled machines one at a time, each time solving a $1|\ r_j\ |Lmax$ problem using a branch-and-bound algorithm. The machine with the greatest *Lmax* is taken as the current bottleneck machine and the corresponding disjunctive arcs are fixed while those on other machines remain unfixed. This procedure also includes a re-optimization process, that is, each time a new machine is scheduled, each of the machines previously scheduled is considered again as an unscheduled machine by deleting the disjunctive arcs that had been fixed before. The corresponding sub-problem is reformulated by re-computing the data necessary and the machine is then rescheduled using the same branch-and-bound

algorithm. It has been found that the re-optimization process usually provides a significant improvement. This shifting bottleneck procedure works well and achieves the optimal solution for a well-known ten job, ten machine benchmark problem in a matter of minutes. A lot of research work has been done on expanding ideas behind the shifting bottleneck procedure, either to improve its performance on the *Jm* | | *Cmax* problem or to adapt it to solve other complex shop scheduling problems.

### 2.5.3 Summary

In summary, the decomposition methods applied in HFS scheduling usually follow the overall procedure as below: (1) problem decomposition, (2) bottleneck identification, (3) sub-problem formulation, (4) solution of the sub-problems and (5) re-optimization. While following the same overall procedure, different researchers differentiate their heuristics in different aspects. For example, some of the researches decompose the problem into stages, while the others decompose the problem into bottleneck stage, upstream sub-problem and downstream sub problem. To decide which way to apply depends on the problem characteristics as well as the solution time/quality tradeoff. There are also different ways to identify the bottleneck selection criteria, the ready times and due dates for sub problems, sub-problem solution procedures and the re-optimization procedures. According to the experiment by Ovacik (1997), the sub-problem solution and re-optimization procedures have a significant effect on both solution quality and computation time, while the bottleneck selection criteria do not have any significant effect as long as a sensible criterion is used.

## 2.6 Genetic Algorithm

Formally introduced in the 1970s by John Holland at University of Michigan, the genetic algorithm is a population-based search and optimization method that simulate the process of natural evolution. It uses techniques inspired by evolutionary biology for example crossover, mutation, inheritance and selection. In the past decade, GA has received considerable attention for solving complicate optimization problems. Hybrid genetic algorithm is a genetic algorithm incorporating other techniques within its framework to produce a hybrid that reaps the best from the combination. Hybrid genetic algorithms have received significant interest in recent years and are being increasingly used to solve real-world problems. Xu (2007) provide a genetic algorithm based on random keys encoding (RKGA) for minimizing makespan on parallel non-identical batching machines. The authors observe that RKGA is very robust and produces consistent solutions across different random seeds within reasonable computation time. Kashan (2008) proposes a hybrid genetic heuristic (HGH) to minimize makespan on parallel batch processing machines with arbitrary job sizes. HGH is characterized by using a robust mechanism for generating initial population, using efficient local search heuristics to bring longer jobs together as a batch and further improve machines load. HGH is claimed to outperform a simulated annealing (SA) approach. Amin-Naseri (2009) develops a three dimensional genetic algorithm (3DGA) to minimize the makespan on hybrid flow shop scheduling with parallel batching. Extensive reviews of GA can be found in Draidi (2004), Chaudhry (2005) and El-Mihoub (2006).

## 2.7 Summary

The problem under study has not received due attention in the literature. Most literature on the scheduling problems in the machine shop environment with batch and discrete machines is focused on two-stage machine environment or three-stage with single machine in each stage. The papers by Amin-Naseri (2009) and Vishwanathan (2007) are the only two publications which are close to the problem under study. Amin-Naseri (2009) conducts the research on multiple stages HFS with parallel uniform batching machines. However, it is assumed that all jobs have identical size, which is different from the assumption in the problem under study. With the identical job sizes assumption, the maximum number of jobs a batching machine can process simultaneously is constant and predetermined by *machine capacity / job size*, while with arbitrary job size assumption the maximum number of jobs in a batch depends on the batch components and cannot be predetermined, therefore increase the difficulty to solve the problem. Vishwanathan (2007) presents a hybrid flow shop scheduling problem with an interior stage consists of parallel batch processing machines and all other stages being discrete-parts processing. It can be considered as a special case of the problem under study, since it assumes identical job sizes, and the batching stage can only locate in between discrete stages.

# 3 PROBLEM DESCRIPTION AND FORMULATION

## 3.1 Problem Characteristics

We are given a set I of stages. Each stage has one or more identical machines in parallel. There is exactly one stage, denoted as *batching stage,* consisting of parallel batch processing machines, while all other stages, denoted as *discrete stage*, consist of discrete machines. The batching stage can be in any position along the flow line. We are also given the set $J$ of jobs. Each job $j \in J$ is described by $\{p_{ji}, s_j\}$ representing its processing time at each stage and job size respectively. Processing times and job sizes are arbitrary and independent to each other. Each job needs to go through all stages in the same order and be processed by only one machine at each stage. In batching stage, machines are able to process two or more jobs simultaneously as long as the total size of jobs in a batch does not exceed the machine capacity. All jobs in a batch start and complete at the same time. The processing time of a batch, defined as $P_b = \max\{p_{ji} \mid j \in b\}$, equals to the longest processing time among all jobs in the batch. The objective is to schedule jobs so that the makespan is minimized.

## 3.2 Mixed-integer Formulation

The sets used in the mathematical formulation are defined below:

$J$ Set of jobs, $j \in J$

$I$ Set of stages, $i \in I$

$Q$ Set of positions in the schedule, $q \in Q$

The parameters used in this mathematical formulation are given below:

$p_{ji}$ Processing time of job $j$ in stage $i$

$C$ Machine capacity

$s_j$ Size of job $j$

$w$ A large number not less than the schedule duration

$n$ Number of jobs

$m_i$ Number of machines in stage $i$

$B$ The position of batching stage

*Decision Variables*

$$x_{ijq} = \begin{cases} 1, & \text{if job } j \text{ is assigned to the } q^{th} \text{ position at stage } i \\ 0, & \text{otherwise} \end{cases}$$

$$y_{iqm} = \begin{cases} 1, & \text{if the job in } q^{th} \text{ position is assigned to machine } m \text{ at stage } i \\ 0, & \text{otherwise} \end{cases}$$

$c_{ji}$ -- The starting time of job $j$ in stage $i$

$C_{\max}$ Makespan or maximum completion time

The mixed-integer formulation developed is as follows:

$$\text{Min } C_{\max} \qquad 3\text{-}1$$

*Subject to:*

$$\sum_q x_{ijq} = 1 \qquad \forall i \in I \ , j \in J \qquad 3\text{-}2$$

$$\sum_m y_{iqm} = 1 \qquad \forall i \in I \ , q \in Q \qquad 3\text{-}3$$

$$\sum_j x_{ijq} <= 1 \qquad \forall i \in I \setminus \{B\} \ , q \in Q \qquad 3\text{-}4$$

$$\sum_j s_j x_{Bjq} \le C \qquad \forall q \in Q \qquad 3\text{-}5$$

$$c_{ji} >= c_{j,(i-1)} + p_{j,(i-1)} \qquad \forall i \in I \setminus \{1, B, B+1\} \ , j \in J \qquad 3\text{-}6$$

$$c_{ji} >= c_{jB} + p_{rB} x_{Brq} - w(1 - x_{Bjq}) \qquad \forall i \in \{B+1\} \setminus \{|I|+1\}, j \in J, r \in J, q \in Q \qquad 3\text{-}7$$

$$c_{ji} >= c_{r,(B-1)} + p_{r,(B-1)} - w(2 - x_{ijq} - x_{irq}) \qquad \forall i \in \{B\} \setminus \{1\}, j \in J, q \in Q, r \in J \qquad 3\text{-}8$$

$$c_{ji} >= c_{ri} + p_{ri} - w(4 - x_{ijq} - y_{iqm} - \sum_{d=1}^{q-1} x_{ird} - y_{irm})$$

$$\forall i \in I, j \in J, r \in J \setminus \{j\}, m \in 1..m_i, q \in Q \setminus \{1\} \qquad 3\text{-}9$$

$$c_{jB} <= c_{rB} + w(2 - x_{Bjq} - x_{Brq}) \qquad \forall j \in J, r \in J \setminus \{j\}, q \in Q \qquad 3\text{-}10$$

$$C_{\max} >= c_{j,|I|} + p_{j,|I|} \qquad \forall j \in J \qquad 3\text{-}11$$

$$x_{ijq} \in \{0,1\} \qquad \forall i \in I \ , j \in J, q \in Q \qquad 3\text{-}12$$

$$y_{iqm} \in \{0,1\} \qquad \forall i \in I, m \in 1..m_i, q \in Q \qquad 3\text{-}13$$

$$c_{ji} >= 0 \qquad \forall i \in I, j \in J \qquad 3\text{-}14$$

The constraints 3-2 and 3-3 are to make sure each job is scheduled exactly once in every stage. 3-4 ensures that no more than one job can be processed at the same time in each discrete machine. 3-5 ensures that the summation of job sizes in a batch does not exceed the machine capacity. 3-6 ensures that a job cannot be started until it is finished in previous stage. 3-7 ensures that at the stage right behind the batching stage, a job cannot be started until all jobs in the same batch it belongs to at previous stage are finished. If job *r* and job *j* are assigned to the same batch *q*, then $x_{Bjq} = x_{Brq} = 1$ and $w(1 - x_{Bjq}) = 0$. The equation is then simplified to $c_{ji} >= c_{jB} + p_{rB} x_{Brq}$. It means job *j* at

stage *B+1* cannot be started until job *r* is completed at stage *B*. On the other hand, if job *j* or job *r* are not assigned to batch *q*, then the constraint is invalid. 3-8 ensures that at the batching stage, a job cannot be started until all jobs in the same batch are released from previous stage. If job *r* and job *j* are assigned to the same batch *q*, then $x_{ijq} = x_{irq} = 1$ and $w(2 - x_{ijq} - x_{irq}) = 0$. The equation is then reduced to $c_{ji} >= c_{r,(B-1)} + p_{r,(B-1)}$. It means job *j* at stage *i* cannot be started until job *r* is completed at stage previous stage. If $x_{ijq} = x_{irq} = 1$ is not true, this constraint is invalid. 3-9 ensures that a job cannot be started unless the jobs scheduled before it on the same machine are finished. Assuming two jobs, *j* and *r*, if job is *r* assigned to the position before *q*, $\sum_{d=1}^{q-1} x_{ird} = 1$, otherwise $\sum_{d=1}^{q-1} x_{ird} = 0$. If job *j* is assigned to the position *q*, and also assigned to the same machine as job *r*, then $x_{ijq} = y_{iqm} = y_{irm} = 1$, $4 - x_{ijq} - y_{iqm} - \sum_{d=1}^{q-1} x_{ird} - y_{irm} = 0$, and the constraints can be reduced to $c_{ji} >= c_{ri} + p_{ri}$, which means that job *j* cannot be started until job *r* is finished. On the other hand, if $x_{ijq} = y_{iqm} = y_{irm} = \sum_{d=1}^{q-1} x_{ird} = 1$ is not true, then the constraint is invalid. 3-10 ensures that all jobs in the same batch start at the same time. 3-11 ensures that makespan is no less than the completion time of any job in the last stage. 3-12, 3-13 and 3-14 are the binary and non-negativity restriction on the decision variables.

Since number of variables, especially number of binary variable is the most important indicator of model complexity, the number of binary variables is used to measure the model complexity in this research. Table 3-1 provides the equation to calculate the number of variables, with |*I*| representing number of stages, |*J*| as number of

jobs and $\boldsymbol{m_i}$ denoting number of machines at stage *i*. To compare the growth rate of binary variables, we use big *O* notation. The growth rate of binary variables is $O(|J|^2)$ based on $|J|$, $O(|I|)$ based on $|I|$), and $O(\sum_{i\in I} m_i)$ based on $m_i$. The table shows that $|J|$, $|I|$ and $m_i$ all affect number of binary variables. Among them, number of jobs $|J|$ has the greatest impact.

Table 3-1 Big O Notation on the Growth Rate of Binary Variables

| Number of Binary Variables | $\|J\|$ | $\|I\|$ | $m_i$ |
|---|---|---|---|
| $\|I\|\|J\|\sum_i m_i + \|I\|\|J\|^2$ | $O(\|J\|^2)$ | $O(\|I\|)$ | $O(\sum_{i\in I} m_i)$ |

This model is validated by manually checking the optimal solution produced by the model. Several problem instances are randomly generated and solved by a commercial solver, AMPL/CPLEX. The optimal schedule obtained by the solver is used to check if all constraints are satisfied. If there is no violation, the model is correct.

## 3.3 Runtime Analysis

### 3.3.1 Experiment Design

The proposed model is implemented on AMPL/CPLEX and run on a desktop computer equipped with a 2.21 GHz *Intel Core 2 Duo* ® processor with 2 GB of RAM. Two experiments are conducted. In the first experiment, the solver is allowed to run for a maximum of half an hour for each problem instance. The percentage of instances whose optimal solution is found within half an hour, denoted by *ROPT*, is calculated using equation 3-15. In the second experiment, the solver is allowed to run until the optimal solution is found. Run time is recorded for each instance.

$$ROPT = \frac{N_{optimal}}{N_{total}} \times 100\% \qquad 3\text{-}15$$

Experiment factors are given in Table 3-2. The number of jobs ($n$), number of stages ($v$) and position of batching stage ($b$) are considered in order to determine if the machine environment and job characteristic have a significant effect on the model performance. The number of machines is fixed at 2 at each batching stage, and 1 or 2 at each discrete stage. The capacity of batching machine is assumed to be 10 in all instances. Job sizes are generated randomly from discrete uniform distribution of [1, 8]. Processing times are generated randomly from discrete uniform distribution of [10, 50]. With these factors considered, there are a total of 18 production scenarios. 10 replicas are generated for each scenario for a total of 180 instances for the first experiment, and 5 replicas for each scenario for a total of 90 instances are tested in the second experiment. Less instances are tested in the second experiment due to the much longer run time it requires.

Table 3-2 Experimental Factors

| Experiment Factors | Levels |
|---|---|
| Number of jobs ($n$) | 4, 5, 6 |
| Number of stages ($v$) | 3, 5 |
| Position of batching Stage ($b$) | Front ,interior, Rear |

### 3.3.2 Computational Results

The main effects plot obtained from the analysis of variance is given in Figure 3-1. The factors $n$, $v$ and $b$ all have a significant effect on *ROPT* with p-value all equal 0.

Figure 3-2 shows the *ROPT* discriminated by *n* and *v*. It also described the average run time the solver spent to find an optimal solution, discriminated by *n*. The percentage of instances solved to optimality within 1800s decreases significantly as *n* or *v* increases. On the other hand, the run time to find the optimal solution increases dramatically as *n* increases. For 4-job problem instances, the average run time is 5.22 minutes. For 5-job problem instances, the average run time is 23.37 minutes. However, for 6-job problem instances, the average run time dramatically increases to almost 2 hour. Clearly, as the problem size increases to beyond 6 jobs, the computational cost of solving the problem to optimality is too large to be used in practical implementations. Figure 3-3 shows that the optimal solution is harder to find when batch processors are located at the interior stage.

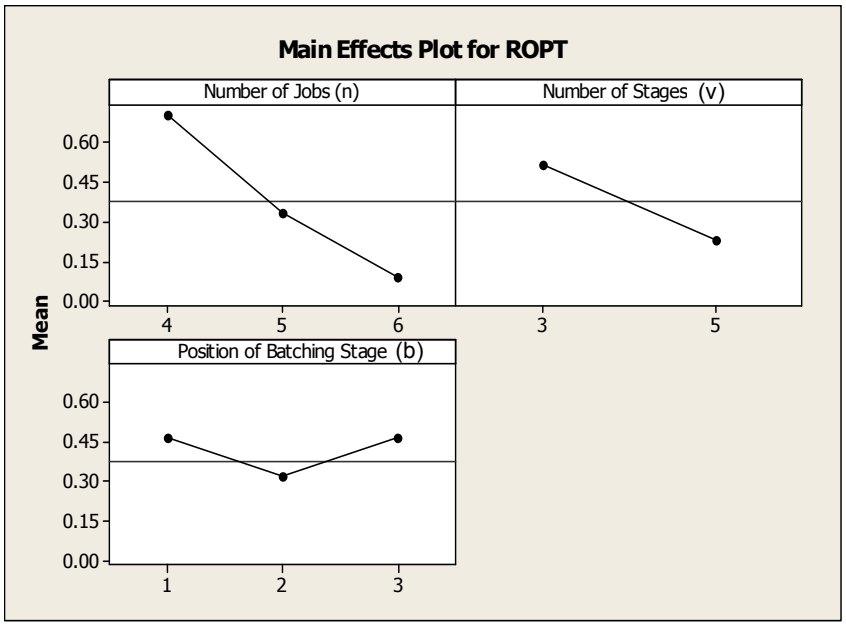



Figure 3-1 Main Effects Plot for Percentage of Instances Solved to Optimality

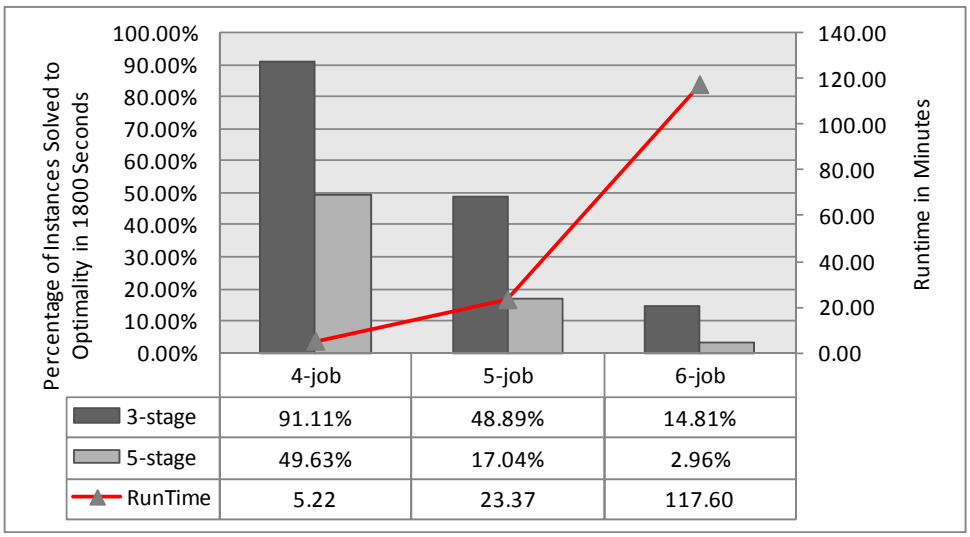



Figure 3-2 Run Time and Percentage of Instances Solved to Optimality

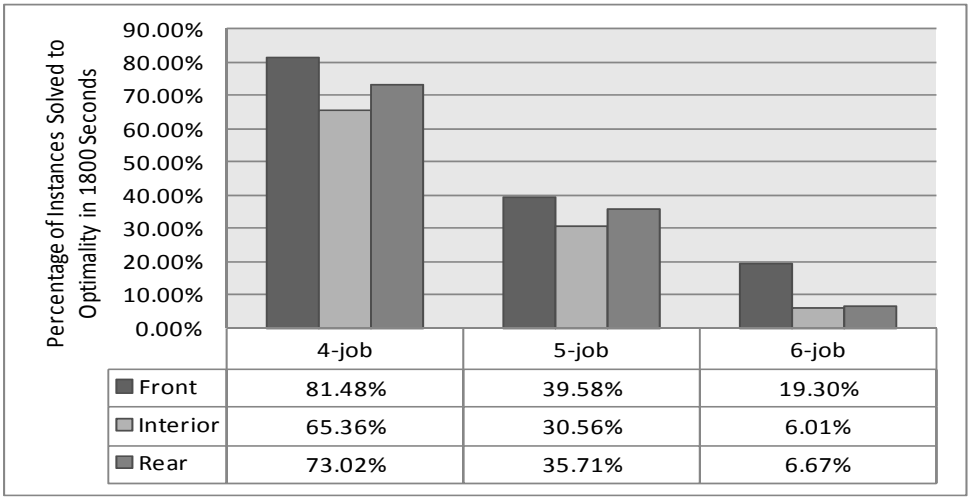



Figure 3-3 Percentage of Instances Solved to Optimality vs. Batching Position

In summary, the number of jobs, the number of stages and the position of batching stage have significant impact on the computational cost. Among these three factors, the number of jobs has the greatest impact. Increase in the number of jobs will dramatically increase computational cost. An instance with more than 6 jobs cannot be

solved to optimality within 2 hours. Therefore, using AMPL/CPLEX to get an optimal solution for the problem under study is computationally expensive and prohibitive for a problem of practical size.

# 4 A LOWER BOUND PROCEDURE

## 4.1 The Proposed Lower Bound Procedure

### 4.1.1 Overview

While a number of lower bounds have been derived for the makespan of parallel machines or the hybrid flow shop scheduling problem, there is no lower bound derived for a scheduling problem associated with hybrid flow shop, parallel batch processing machines and jobs of arbitrary sizes. The lower bound procedure derived in this dissertation incorporates several well known lower bounds which are originally developed for parallel discrete processing machines, parallel batch processing machines or hybrid flow shops with discrete processing machines, and enable to calculate a tight lower bound for the problem under study.

Figure 4-1 shows the main steps in the proposed lower bound procedure. First of all, the jobs in the batching stage are grouped into batches. Based on this, the existing lower bound procedures for parallel discrete-processing machines can be used to calculated the single-stage lower bound for the parallel batch processing machines, with the job processing times replaced with batch processing times. And then the global lower bound for the whole shop is calculated based on the single-stage lower bounds.

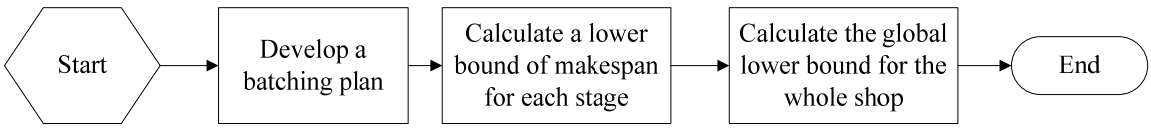



Figure 4-1 Main Steps of the Proposed Lower Bound Procedure

The proposed three-step lower bound procedure is as follows:

1) Develop a batching plan: A batching scheme proposed by Kashan et al. (2008) is applied. Based on Kashan's batching scheme, a bin packing procedure is used to further improve the batching plan.
2) Derive a lower bound for each stage: A lower bound for the makespan of each stage is derived by the scheme proposed by Webster (1996). The lower bound is furthered improved by the machine-subset lifting procedure and the bin packing procedure. At this step, the batch processing stage is treated as a discrete stage by replacing job processing times with batch processing times.
3) Calculate a global lower bound: two lower bounds for the whole shop are calculated using two lower bound schemes, Machine-based Lower Bound (MBLB) proposed by Santos (1995) and Stage-based Lower Bound (SBLB) proposed by Kurz and Askin (2003) . The lower bound with the greater value is chosen as the global lower bound. For small size instances the global lower bound can be further enhanced by a job-subset lifting procedure.

### 4.1.2 Batching Plan

#### 4.1.2.1 Batching Plan Proposed by Kashan et al. (2008)

Kashan et al. proposed that a lower bound of the makespan on the parallel batching machines can be calculated by allowing jobs to be split and processed in different batches. This is done by constructing an instance where each job $j$, of the original problem, is replaced with $s_j$ jobs of unit size, of which processing times all equal

$p_j$ . The obtained unit-size jobs are sorted in the LPT order. Starting from the top of the list, successively group every *S* adjacent jobs into a batch. This batching procedure is called LPT-Full batching, and results in the formation of a set *B* of batches, with $|B| = \left\lceil \frac{1}{S} \sum_{j \in J} s_j \right\rceil$. Based on this, Kashan et al. further improved it by separating the big jobs from being spit. Let *J* be the original set of jobs, and let $J_1$ be the set of jobs in *J* that satisfy equation 4-1.

$$J_1 = \left(j \in J | S - s_j < min_{k \in J}\{s_k\}\right) \qquad 4\text{-}1$$

The rest of the jobs in J are assigned to set $J_2$. If $j \in J_1$ assigned to a batch, the residual capacity of the batch ($S - s_j$) is smaller than the smallest job. Consequently, all the jobs in set $J_1$ are large jobs and will not accommodate any other job in the same batch. Each job from $J_1$ is therefore separately assigned to a unique batch. On the other hand, each job from $J_2$ is split into unit-sized jobs. Applying the LPT-Full batching rule, a set *B* of batches is formed with $|B| = |J_1| + \left\lceil \frac{1}{S} \sum_{j \in J_2} s_j \right\rceil$. With this modification, the number of batches formed might increase and therefore lead to an increment in the lower bound.

#### 4.1.2.2 Improvement: Bin Packing Procedure

A Bin Packing Procedure (*BPP*) is applied here to further improve the batching plan. The mechanism of *BPP* is to calculate the minimum number of batches required *(L)* to contain all the jobs. If $L > |B|$, then $L - |B|$ amount of batches should be added to the former batching plan *B*.

For the set of job sizes $\{s_1, \dots, s_n\}$ , let $W$ be the set of all distinct values $s_j < \frac{S}{2}$. For each integer $s \in W$, let

$$J_1(s) = \left(j \in J: s_j > S - s\right) \quad 4\text{-}2$$

$$J_2(s) = \left(j \in J: \frac{S}{2} < s_j \leq S - s\right) \quad 4\text{-}3$$

$$J_3(s) = \left(j \in J: s < s_j \leq \frac{S}{2}\right) \quad 4\text{-}4$$

Define $L_\alpha(s)$ by

$$L_\alpha(s) = |J_1(s)| + |J_2(s)| + max\left(0, \left\lceil \frac{\sum_{j \in J_3(s)} s_j - (|J_2(s)|S - \sum_{j \in J_2(s)} s_j)}{S} \right\rceil\right) \quad 4\text{-}5$$

The number of batches required can be decided by:

$$L = max_{s \in W} L_\alpha(s) \quad 4\text{-}6$$

If $L > |B|$, let $|B| = L$ by adding $L - |B|$ batches to the former batching plan $B$. The processing times of the new added batches are set as the the $L - |B|$ shortest job processing times. After batches are formed, let $P_b$ be the processing time of $b \in B$ batch. List the batches in $B$ in non-increasing order of their processing times such that $P_1 \geq P_2 \geq \cdots \geq P_{|B|}$. $P_b$ will be used to calculate the lower bound for the batching stage and the whole shop. The *BPP* might increase the number of batches, and therefore leads to an increment in the lower bound.

4.1.2.3 The Procedure to Get the Batching Plan

The procedure to get the batching plan for deriving the lower bound is given in Figure 4-2.

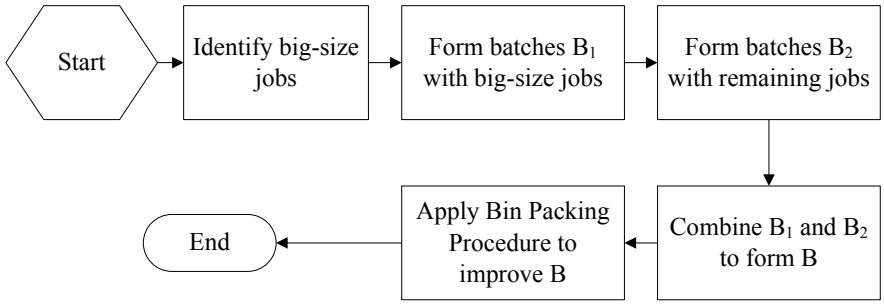



Figure 4-2 Batching Procedure for Lower Bound

1) Identify big-size jobs: Find out all the big-size jobs $J_1$ which cannot accommodate any other job in the same batch using the equation 4-1.
2) Form batches with big-size jobs: Form a set of batches $|B_1|$ by putting each job of $J_1$ into one batch.
3) Form batches with remaining jobs: Split every remaining job $j$ into $s_j$ unit-size jobs with processing times equal to $p_j$, thus form a job set $J_2$. Sort $J_2$ in the LPT order and successively group the $S$ jobs with longest processing times into the same batch to form a second set of batches $B_2$, with $|B_2| = \left\lceil \frac{1}{S} \sum_{j \in J_2} s_j \right\rceil$.
4) Combine $B_1$ and $B_2$ to obtain all batches $B$ and corresponding batch processing times $P_b$ sorted by LPT rule.
5) Apply the bin packing procedure described in 4.1.2.2 to improve the batching plan.

Among this procedure, step 1) to 4) is the same as the procedure proposed by Kashan et al., and 5) is proposed by author of this dissertation.

### 4.1.3 Single-stage Lower Bound

#### 4.1.3.1 A Trivial Lower Bound

Let $P = \{p_1, \ldots, p_n\}$ be a problem instance where $p_i$ is the processing time of job j and jobs are indexed such that $p_1 \geq p_2 \geq \cdots \geq p_n$. Assume the number of machines in parallel is $m$ and all machines are available at time 0. A trivial lower bound for the scheduling problem P//Cmax can be derived by using following equation by Baker (1974):

$$plb0 = max\left(p_1, \left\lceil \frac{\sum_j p_j}{m} \right\rceil, p_m + p_{m+1}\right) \quad 4\text{-}7$$

The first term in 4-7 is built on machine capacity relaxation. It implies that the makespan has to be at least the largest processing time of all the jobs. The second term is based on the preemption relaxation. It allows the total processing time of all the jobs equally divided among all the $m$ machines. The third terms implies that if there are at least $m$+1 jobs in $J$, the makespan has to greater than or equal to the sum of the $m^{th}$ and $(m+1)^{th}$ smallest processing times.

#### 4.1.3.2 An Enhanced Lower Bound

An enhanced lower bound for P//Cmax based on Webster's general lower bound (1996) is used to obtain a tighter lower bound. Webster's general lower bound is originally used for parallel discrete processing machines. A modification is applied in this dissertation to accommodated batch processing.

In Webster's lower bound derivation, for $P = \{p_1, \ldots, p_n\}$, with $p$ sorted in the non-increasing order, let $P_\rho$ be a relaxed instance defined as 4-8.

$$P_\rho = \{p_1 + \cdots + p_{n'}, \rho, \ldots, \rho\} \quad 4\text{-}8$$

Where $\rho = \gcd(p_1, \ldots, p_n)$

$n' \leq n$, and $n'$ is the largest integer that satisfies at least one of the following conditions:

*1)* $p_j/p_{j+1}$ *is integer for* $j = 1, \ldots, n' - 1$ 4-9

2) $p_{m-1} \geq p_m + \cdots + p_{n'-2}$ 4-10

3) $n' \leq 2m$ and $p_{2m-n'+1} \leq 2p_{n'}$ 4-11

The minimum makespan of $P_\rho^c$, can be found using the LPT rule, and denoted as $W_O(P_\rho)$.

For batch processing stage, to apply Webster's lower bound scheme, each batch is treats as a job, and job processing times are replaced with batching processing times.

Taking into acount the trivial lower bound 4-7, a tighter lower bound for parallel machines scheduling problem can be obtained by using following equation:

$$plb = max(W_O(P_\rho), p_1, p_m + p_{m+1}) \quad 4\text{-}12$$

4.1.3.3 Improvement 1: Machine Sub-set Lifting Procedure

A machine sub-set lifting procedure proposed by Haouari (2004) is used to improve the lower bound quality. According to the lemma proposed by the authors, *in any feasible schedule of a parallel machine problem with n jobs and m machines, there is at least a set of k machines (*$1 \leq k \leq m$*), which must process at least* $\lambda_k$ *jobs, where*

$$\lambda_k = k\left\lfloor\frac{n}{m}\right\rfloor + min\left(k, n - \left\lfloor\frac{n}{m}\right\rfloor m\right) \quad 4\text{-}13$$

Therefore a reduced problem can be constructed with $k$ machines in parallel ($1 \leq k \leq m$) and $\lambda_k$ jobs of $J$ with smallest processing times. Let $plb^k$ denotes a lower bound for the reduced instance $\lambda_k$, a lower bound can be derived from following equation:

$$plb = max_{1\leq k\leq m} plb^k \quad 4\text{-}14$$

4.1.3.4 Improvement 2: Bin Packing Procedure

Martello (1990) derived a lower bound for P||Cmax by continuously checking whether the *n* jobs could be processed on the *m* machines such that the makespan does not exceed a trial value *C*. If the number of machines required exceeds *m*, the lower bound is then increased by one unit: *C=C+1*. This idea is applied here to improve *plb*.

Let *V* be the set of all distinct values $p_j < \frac{C}{2}$. For each integer $p \in V$, let

$$J_1(p) = \left(j \in J: p_j > C - p\right) \quad 4\text{-}15$$

$$J_2(p) = \left(j \in J: \frac{C}{2} < p_j \leq C - p\right) \quad 4\text{-}16$$

$$J_3(p) = \left(j \in J: p < p_j \leq \frac{C}{2}\right) \quad 4\text{-}17$$

Define $L_\alpha(p)$ by

$$L_\alpha(p) = |J_1(p)| + |J_2(p)| + max\left(0, \left\lceil \frac{\sum_{j\in J_3(p)} p_j - (|J_2(p)|C - \sum_{j\in J_2(p)} p_j)}{C} \right\rceil\right) \quad 4\text{-}18$$

Number of machines required can be decided by:

$$L = max_{p\in V} L_\alpha(p) \quad 4\text{-}19$$

If $L > m$, let $C = C + 1$, and repeat above procedure until $L \leq m$. The final value of $C$ is a valid lower bound for the $P||Cmax$.

4.1.3.5 The Procedure to Calculate Single-stage Lower bound

Based on above discussion, a procedure to calculate the single-stage lower bound is shown in Figure 4-3 and presented as below:

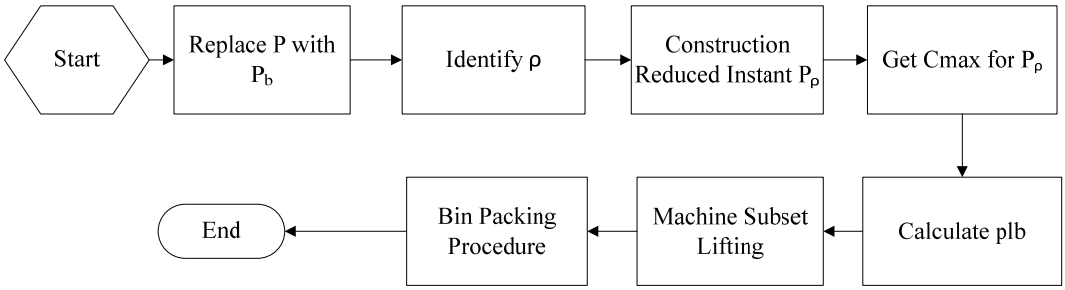



Figure 4-3 The Procedure to Calculate plb

1) If the stage is the batching stage, replace the original processing times $P$ with batch processing times $P_b$.
2) Set $\rho = \gcd(p_1, \ldots, p_n)$
3) Construct a reduce problem instance $P_\rho$ using equation **4-8**, 4-9, 4-10 and 4-11.

4) Apply the LPT rule on the reduced instance $P_\rho$ to obtain the optimal makespan, $W_O(P_\rho)$.

5) Calculate the single stage lower bound, $plb$, using equation 4-12.

6) Apply machine subset lifting procedure to improve $plb$.

7) Apply bin packing procedure to further improve $plb$.

### 4.1.4 Global Lower Bound

#### 4.1.4.1 A Trivial Global Lower bound

A trivial lower bound of makespan for the problem under study is given in the equation 4-20.

$$GLB0 = max_{j\in J}\left\{\sum\nolimits_{i\in I} p_{ji}\right\} \qquad 4\text{-}20$$

It is based on the relaxation of machine capacities in all stages by assuming machine number ≥ job number in each stage. This trivial lower bound will be combined with lower bounds derived in the rest of this chapter to obtain a final lower bound.

#### 4.1.4.2 Stage-based Lower bound Scheme

The stage-based lower bound is in the method proposed by Kurz and Askin (2003) and Kashan (2008). A lower bound of the whole shop is developed based on a single stage and the greatest stage-based bound is the bound which can be used for the

entire problem. The stage-based bounds are denoted as $lb1_i$, and the global lower bounds are denoted as *GLB1*. A global lower bound *GLB1* can be derived as follows:

$$GLB1 = max_{i \in I}\{lb1_i\} \quad 4\text{-}21$$

Where

$$lb1_i = min_{j \in J}\left\{\sum_{h=1}^{i-1} p_{jh}\right\} + plb_i + min_{j \in J}\left\{\sum_{h=i+1}^{|I|} p_{jh}\right\} \quad 4\text{-}22$$

$plb_i$ is the lower bound for the makespan at stage *i*. Stage *i* could be batch processing stage or discrete stage. The procedure to calculate $plb_i$ is stated in 4.1.3.

4.1.4.3 Machine-based Lower Bound Scheme

A lower bound originally proposed by Santos (1995) is applied to calculate the machine-based lower bound.It is based on the concept of finding the mean of the machine-based lower bounds. This scheme is originally used for hybrid flow shops with discrete processing machines. In this dissertation, it is modified to accommodate batch processing.

A machine-based lower bound is calculated by using following equations:

$$GLB2 = max\{lb2_i\} \quad 4\text{-}23$$

Where

$$lb2_i = \frac{1}{m_i}\left[\sum_{v=1}^{m_i} JL_{vi} + \sum_{j=1}^{n} p_{ji} + \sum_{v=1}^{m_i} JR_{vi}\right] \quad 4\text{-}24$$

$$lb2_i = \frac{1}{m_i}\left[\sum_{v=1}^{m_i} JL_{vi} + \sum_{j=1}^{n} p_{ji} + \sum_{v=1}^{m_i} JR_{vi}\right] \quad 4\text{-}25$$

$JL_{li}$: *the lth smallest value of* $\sum_{k=1}^{i-1} p_{jk}$, $\forall j \in J$, *for a fixed* $i$ .

$JR_{li}$: *the lth smallest value of* $\sum_{k=i+1}^{|I|} p_{jk}$, $\forall j \in J$ , *for a fixed* $i$ .

$m_i$: *the number of machines at stage* $i$

To modify Santo's lower bound scheme to accommodate batch processing, $p_{ji}$ is replaced by $P_b$ at batch processing stage, and $P_b$ can be obtained using the batching plan introduced in 4.1.2.

#### 4.1.4.4 The Global Lower Bound

The global lower bound *GLB* is determined by the maximum value among the trivial lower bound *GLB0*, the stage-based lower bound *GLB1* and the machine-based lower bound *GLB2*.

$$GLB = max\{GLB0, GLB1, GLB2\} \quad 4\text{-}26$$

### 4.1.5 A Lifting Procedure

A job sub-set lifting procedure proposed by Carlier and Pinson (1998) can find a tighter lower bound by considering different subsets of *J*. Define $J_k \subseteq J$ as the subset of jobs that contains any *k* jobs from *J*. Let $GLB^k$ denotes a lower bound for the reduced instance $J_k$, a valid lower bound can be found as follows:

$$GLB = max_{min_{i\in I} m_i \le k \le n} GLB^k \quad (4\text{-}27)$$

Where

$min_{i\in I} m_i$ is the minimum number of machines in parallel among all stages.

Note that it is only necessary to consider the values of $k > min_{i\in I} m_i$, since the derived lower bound is dominated by the trivial bound $max_{j\in J}\{\sum_{i\in I} p_{ji}\}$.

### 4.1.6 The Global Lower Bound Procedure

The overall procedure to calculate the lower bound for the makespan on the problem under study is given in Figure 4-4 and presented as follows:

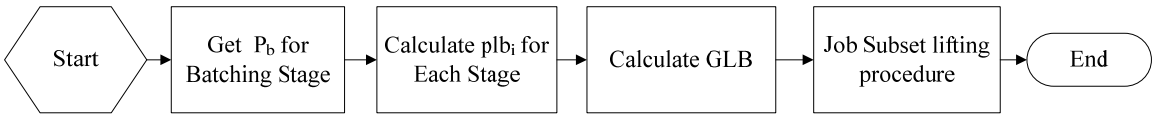



Figure 4-4 The Procedure to Calculate GLB

1) For the batching stage, get the batching plan using the method introduced in 4.1.2 to obtain $P_b$.
2) Calculate the single-stage lower bounds $plb_i$ for each stage $i$ as introduced in 4.1.3. If $i$ is the batching stage, replace the original processing times in stage i with $P_b$.
3) Calculate the trivial global lower bound *GLB0* using equation 4-20. Calculate the stage-based lower bound *GLB1* using equations 4-21 and 4-22. Calculate the machine-based lower bound *GLB2* using equations and 4-23, 4-24 and 4-25. Determine the global lower bound using equation 4-26.

4) Apply job subset lifting procedure introduced in 4.1.4.4 to improve *GLB*.

## 4.2 Qualification of the Proposed Lower Bound

### 4.2.1 Experiment Design

Computational experiments are conducted to evaluate the quality of the proposed lower bound. First of all, the proposed lower bound for parallel batching machines, *plb*, is compared with the lower bound derived by Kashan (2008). And then *GLB*, the proposed global lower bound for the problem under study, is first compared with the makespan solved by CPLEX, and then evaluated against the lower bound proposed by Amin-Naseri (2009).

### 4.2.2 Comparing *plb* with the Lower Bound by Kashan et al (2008)

The lower bound of the makespan on parallel batching machines, *plb*, is derived in 4.1.3, and is a component of the global lower bound *GLB*. The quality of *plb* is compared with the quality of lower bound derived by Kashan. Kashan 's lower bound is calculated using following equation.

$$C_{max}^{LB} = max\left\{max_{j\in J}\{p_j\}, \frac{\sum_{b\in B} P_b}{m}, P_m + P_{m+1}\right\} \quad 4\text{-}28$$

Where

$P_b$ is the batch processing time of $b \in B$.

`$P_1 \geq P_2 \geq \cdots \geq P_{|B|}$

The experimental factors are given in Table 4-1. A total of 1200 instances are tested. The average improvement of *plb* over Kashan's lower bound is calculated by following equation:

$$\%IM = \frac{plb \; - \; Kashan's\ Lower\ Bound}{Kashan's\ Lower\ Bound} \times 100\% \qquad 4\text{-}29$$

Table 4-1 Experimental Factors

| Factors | Level | Factors | Level |
|---|---|---|---|
| *n* | 10, 20, 50, 100, 200 | *m* | 3, 5 |
| *MaxS* | 5, 20 | *MaxP* | 10, 30 |
| Machine Capacity | 20 | | |

The job size *s* is generated from the discrete uniform [1, *MaxS*], and the processing time *p* is generated from the discrete uniform [1, *MaxP*]. Machine capacity is fixed at 20, and machine number is 2 or 4.

The comparison result reveals that *plb* dominates Kashan's lower bound and results in a 0.42% improvement in average for all instances tested.

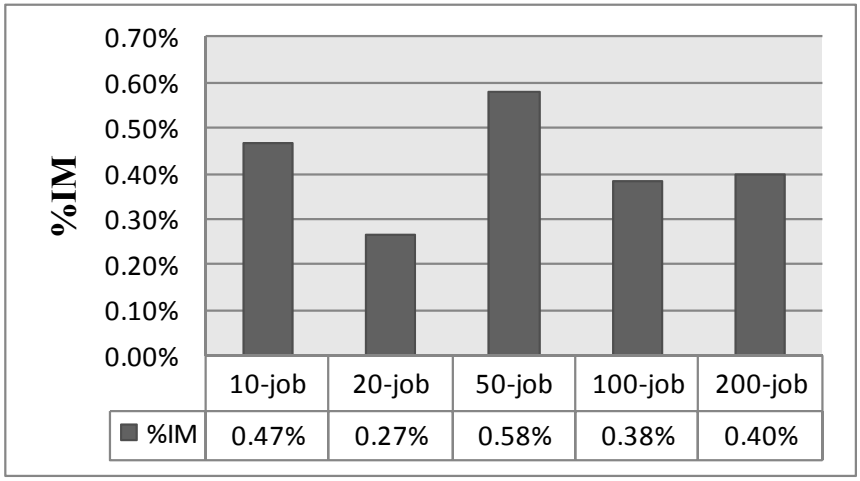



Figure 4-5 Percentage of Improvement of plb Over Kashan's Lower Bound

### 4.2.3 Comparing GLB with the Optimal Makespan Solved by CPLEX

The lower bound is compared with the makespan of the optimal solution solved by CPLEX, denoted as $C_{max}^{OPT}$. The relative gap between the lower bound and the optimal makespan, as defined in following equation, is used to evaluate the lower bound tightness.

$$RDEV^{OPT-GLB} = \frac{C_{max}^{OPT} - GLB}{C_{max}^{OPT}} \times 100\% \qquad 4\text{-}30$$

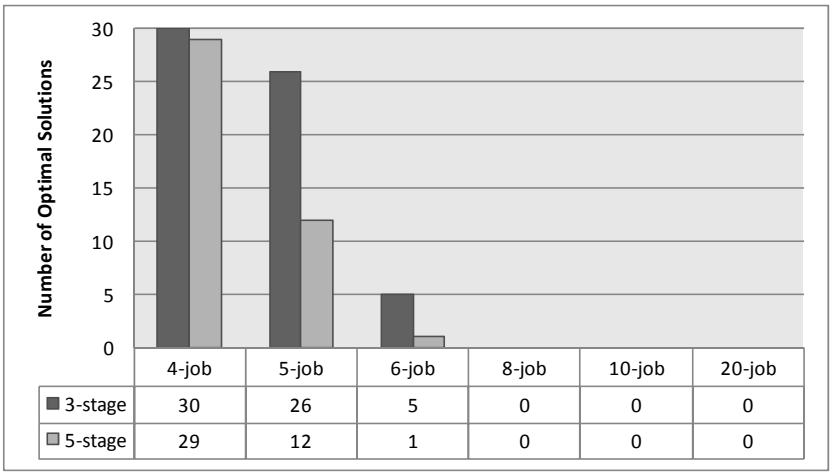
The experiment data are randomly generated based on two factors:

*Number of Jobs (n):* 4, 5, 6, 8, 10, 20

*Number of Stages (v):* 3, 5

The number of machines at the batching stage is fixed at 2 and the number of machines at discrete stages is either one or two. The capacity of batching machine is assumed to be 10 in all instances, and the size of job is generated from discrete uniform distribution of [1, 8].

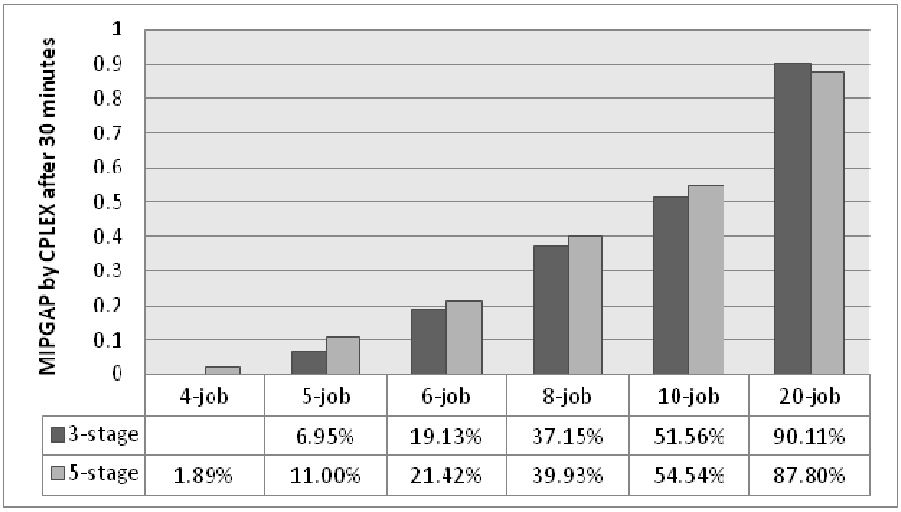
With two factors considered, there are a total of 12 scenarios and 30 problem instances are generated for each scenario resulting in 360 problem instances in total.




| | 4-job | 5-job | 6-job | 8-job | 10-job | 20-job |
|---|---|---|---|---|---|---|
| 3-stage | 30 | 26 | 5 | 0 | 0 | 0 |
| 5-stage | 29 | 12 | 1 | 0 | 0 | 0 |

Figure 4-6 Number of Instances solved to the Optimality by CPLEX in 30 Minutes

Among 360 randomly generate instances, optimal solutions are found within 30 minutes by CPLEX for 103 instances. The distribution of instances solved to the optimality by CPLEX is presented in Figure 4-6. The optimal solution is found for most of the 4-job instances, and 5-job instances with 3 stages. Among instances with 5 jobs and 5 stages, 12 instances out of 30 are solved to the optimality. For 6-job instances with 3 stages and 5 stages, optimal solution is found for only 5 out of 30 instances and 1 out of 30 instances, respectively. None of the 8-job, 10-job and 20-job instances is solved to the optimality. For all the instances whose optimal solution are found within 30 minutes, the relative gap between the linear relaxation and the best solution found, *MIPGAP*, are given in **Error! Reference source not found.**. It shows that the gap increases substantially when the number of jobs increases. For 10-job instances and 20-job instances, the gaps exceed 50%, indicating the poor quality of the solution found by CPLEX.




| | 4-job | 5-job | 6-job | 8-job | 10-job | 20-job |
|---|---|---|---|---|---|---|
| 3-stage | | 6.95% | 19.13% | 37.15% | 51.56% | 90.11% |
| 5-stage | 1.89% | 11.00% | 21.42% | 39.93% | 54.54% | 87.80% |

Figure 4-7 Average MIPGAP for Instances Not Solved to the Optimality

Figure 4-8 shows the average *RDEV* of lower bounds from the optimal makespan for all 103 instances discriminated by number of jobs and stages. The average *RDEV* for all 103 instances is 6.20%. Among 4-job instances, the *RDEV* is 5.43% and 6.35% for 3-stage and 5-stage instances, respectively. Among 5-job instances, the *RDEV* is 7.52% and 5.82% for 3-stage and 5-stage instances. While among 6-job instances, the *RDEV* is 5.19% and 0% for 3-stage and 5-stage instances. Please note that only one instance is included in the 6-job/5-stage category. Therefore, this category should be discarded. The result reveals that the proposed lower bound is consistently close to the optimal makespan, and the number of stages and number of jobs show no great impact on lower bound quality.

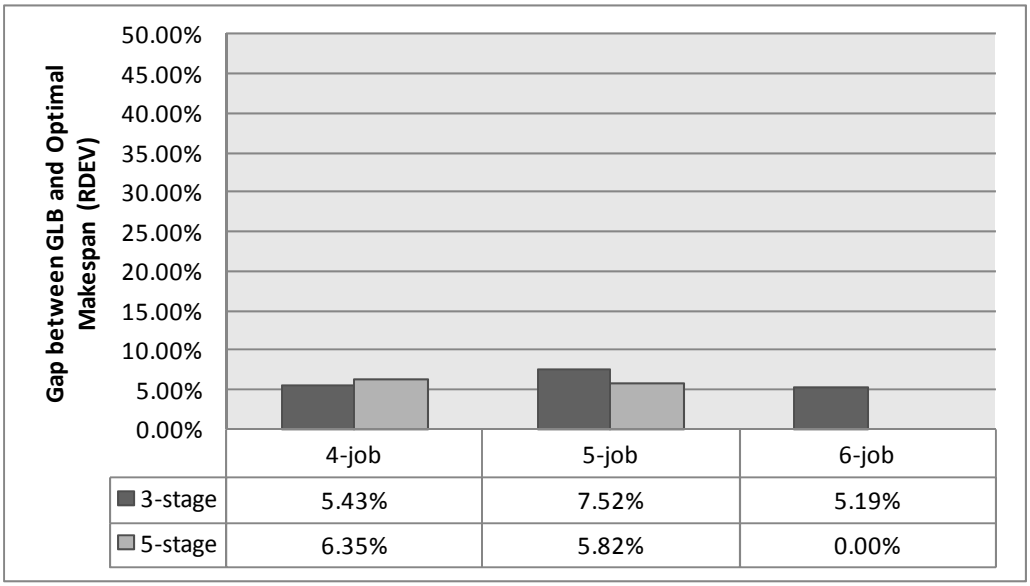



Figure 4-8 Relative Deviation of GLB from the Optimal Makespan

For the remaining instances whose optimal solution is not found by CPLEX within 30 minutes, the lower bound is compared with the best integer solution found, and

the relative gap, $RDEV^{BS-GLB}$, is calculated using the equation 4-29. The value of $RDEV^{BS-GLB}$ is given in Figure 4-9. The relative gap between GLB and the best integer solution found is significantly larger than the gap between GLB and the optimal solution, and the gap increases substantially when the number of jobs or number of stages increases. This can be explained by the poor quality of the CPLEX solution for larger instances. As we can see in **Error! Reference source not found.**, the best integer solution found is deviated from the linear relaxation solution when the problem size increases.

$$RDEV^{BS-GLB} = \frac{C_{max}^{OPT} - GLB}{C_{max}^{OPT}} \times 100\% \qquad 4\text{-}31$$

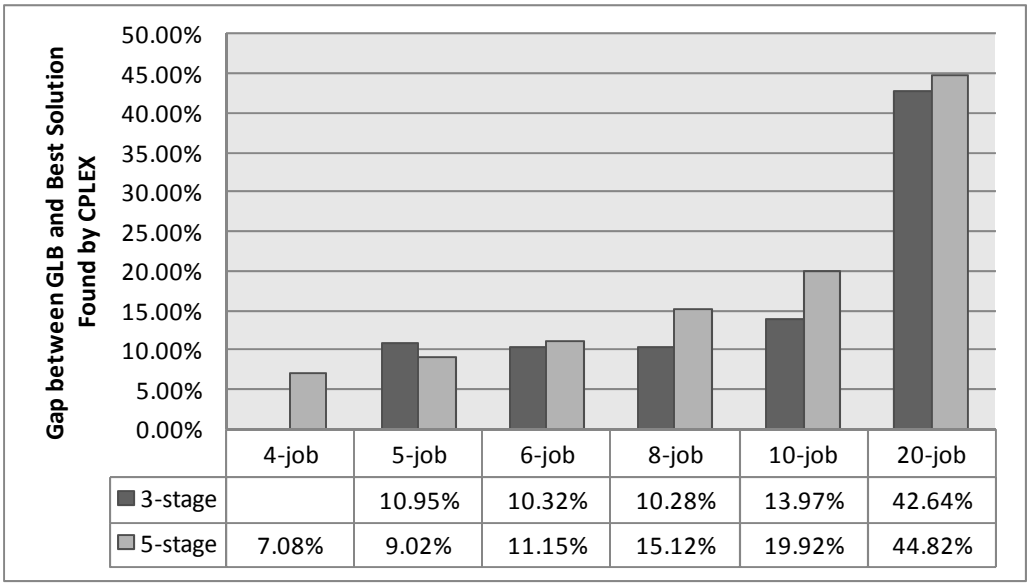



Figure 4-9 Relative Deviation (RDEV) of GLB

### 4.2.4 Comparing *GLB* with Amin-Naseri's Lower Bound (*ALB*)

Amin-Naseri (2009) calculate a lower bound for the makespan on the hybrid flow shop with parallel batching machines, of which machines are uniform in each stage and job sizes are identical. Since the problem the author studied is similar, with a few modifications, Amin-Naseri's lower bound procedure can be used to calculate the lower bound for our problem. Machine speeds are set to one, and arbitrary job sizes are added. The modified Amin-Naseri's lower bound is calculated using following equations:

$$ALB = max_{j\in J, i\in I}\left\{lb_i, max_{j\in J}\left\{\sum_{i\in I} p_{ji}\right\}\right\} \quad \text{4-32}$$

Where

$$lb_i = \sum_{h=1}^{i-1} min_{j\in J}\{p_{jh}\} \quad \text{4-33}$$

$$+max\left\{\frac{\sum_{j\in J}(p_{ji}\times s_j)}{m_i\times C}, max_{j\in J}\{p_{ji}\}\right\} \quad \text{if i is the batching stage}$$

$$+max\left\{\frac{\sum_{j\in J} p_{ji}}{m_i}, max_{j\in J}\{p_{ji}\}\right\} \quad \text{if i is the batching stage}$$

$$+\sum_{h=i+1}^{[I]} min_{j\in J}\{p_{jh}\}$$

$m_i$ is the number of parallel machines in stage *i*.

*C* is the capacity of batch processing machines.

*ALB* and the proposed lower bound *GLB* are tested on 1458 randomly generated instances. The average improvement of *GLB* over *LB* is calculated by following equation:

$$\%IM = \frac{GLB - ALB}{ALB} \times 100\% \qquad 4\text{-}34$$

The comparison result reveals that *GLB* dominates *ALB* and results in a 3.18% improvement in average for all instances tested. The percentage of improvement discriminated by number of jobs is given in Figure 4-10. Among all 1458 instances, *GLB* outperforms *LB* for 965 instances, while *LB* outperforms *GLB* for 0 instance.

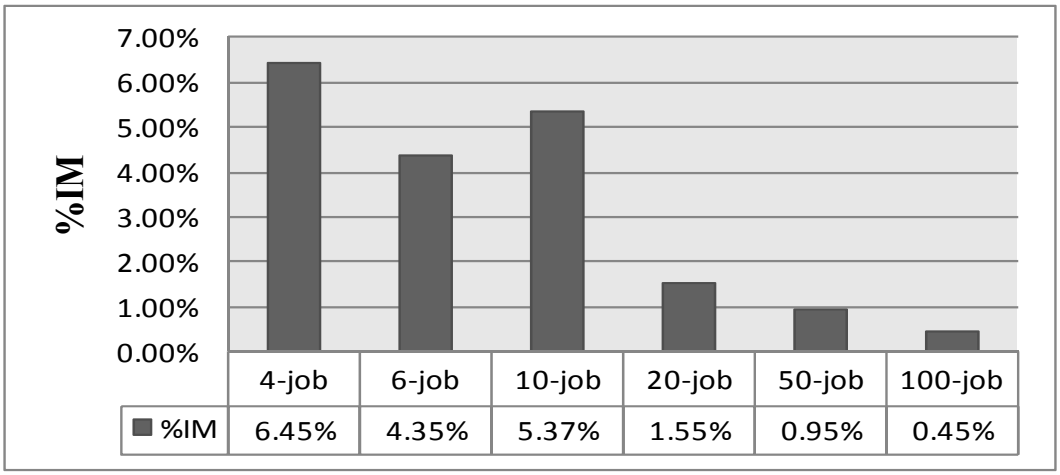



Figure 4-10 Relative Improvement of GLB on Amin-Naseri's Lower Bound

### 4.2.5 Summary

Based on the experiment results, a conclusion can be made that the proposed lower bound procedure generates tight lower bound for the problem under study. The lower bound quality remains consistent when the problem size increases. The derived lower bound dominates the lower bound proposed by Amin-Naseri, and the derived lower

bound on parallel batching machines dominates the lower bound proposed by by Kashan et al (2008).

# 5 PROPOSED HEURISTIC APPROACH

A bottleneck-first-decomposition heuristic (BFD) is proposed for the problem under study in this chapter to overcome the computational (time) problem encountered while using the commercial solver for analytic solutions.

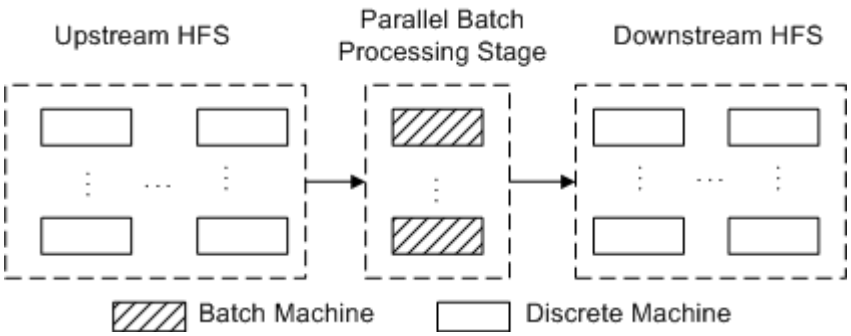



Figure 5-1 Problem Decomposition

## 5.1 Overview of the Proposed Heuristic Approach

The proposed BFD heuristic is inspired by the Theory of Constraint and decomposition approaches, and follows the basic procedure of the shifting bottleneck heuristic designed by Adams (1988) and improved by Balas (1995). Even though shifting bottleneck is originally designed for job shops, some literature has applied it to flow shops, for example Cheng et al. (2001). To reduce computational time, instead of treating a single-stage scheduling as a sub-problem, BFD decomposes the problem into three sub-problems: parallel batch processing machines scheduling problem (PB), upstream hybrid flow shop scheduling problem (UHFS) and downstream hybrid flow shop scheduling problem (DHFS) (Figure 5-1). In case that batch processing machines lie at the first stage, UHFS doesn't exist, while when PB lies at the last stage, DHFS no longer exists.

The original problem $FF|batch1, s_j|Cmax$ is referred to as master problem. The heuristic schedules one sub-problem at a time until all sub-problems have been scheduled. A re-scheduling process is also included in the heuristic: every time a sub-problem is scheduled, all previously scheduled sub-problems need to be re-scheduled.

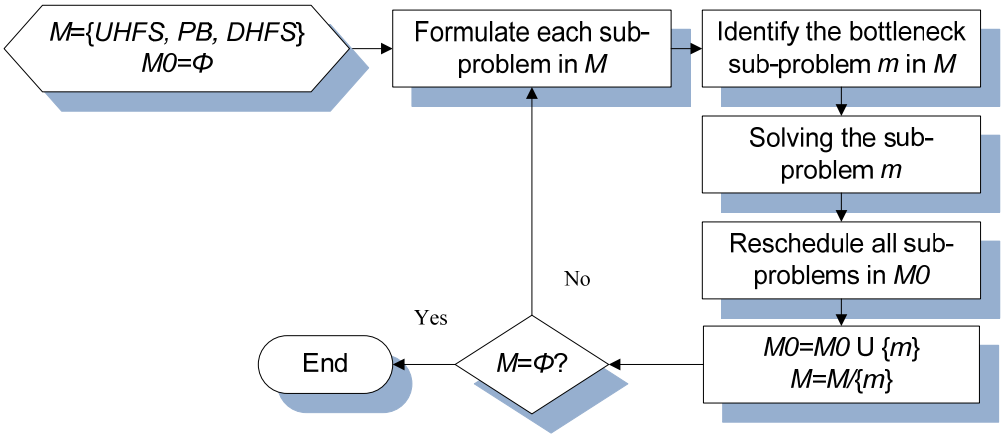



Figure 5-2 Major Steps of BFD

The major steps of BFD are given in Figure 5-2, and is described as below:

Step 1: Let *M* be the set of all sub-problems and *M0* be the set of sub-problems which have been scheduled. Set *M0*=Φ.

Step 2: Formulate each un-scheduled sub-problem in *M*.

Step 3: Identify a bottleneck sub-problem $m \in M$.

Step 4: Solving sub-problem *m*.

Step 5: Re-schedule all sub-problems in *M0* based on the schedule of *m*.

Step 6: Set $M0 = M0 \cup \{m\}$ and $M = M \backslash \{m\}$.

Step 7: If $M = \Phi$, stop. Otherwise, go to step 2.

The BFD heuristic is applied forwardly and backwardly on the same problem instance, and the better solution is reported as the final solution.

Although the shifting bottleneck procedure is a common basic procedure, each step can be accomplished in different ways. In the remaining part of this chapter, each major step will be discussed in detail.

The BFD heuristic consists of a main procedure and several sub-procedures. The map of the main procedure and sub-procedures are shown in Figure 5-3. The sub-procedure, Bottleneck Identification and Scheduling (BIS), is called by the main BFD procedure iteratively to identify and schedule the bottleneck sub-problem. When scheduling the bottleneck sub-problem, two procedures, List-scheduling-with-delay (LSD) and Bottleneck Approach based on Jackson's Heuristic (BA-Jackson), are called by BIS to solve the sub-problems. Additionally, BA-Jackson also has a sub-procedure, Modified Jackson's heuristic, to identify and schedule the bottleneck stage of the sub-problem. The details of these procedures are provided in 5.4-5.6.

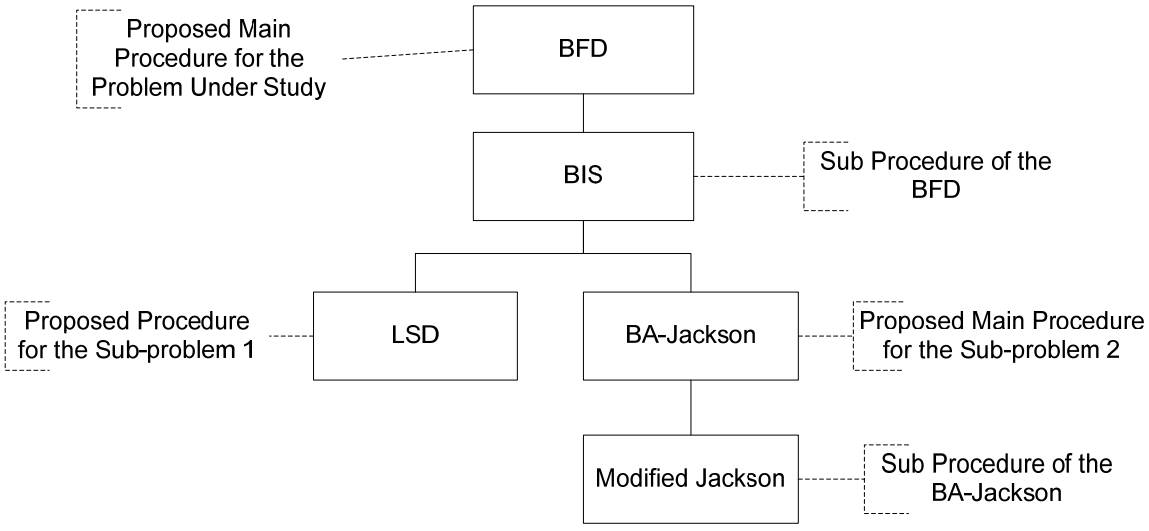



Figure 5-3 Map of the Heuristic Procedures

## 5.2 Sub-problem Formulation

Sub-problems have to be formulated in such a way that interactions between sub-problems should be formed appropriately, so as to facilitate the search for a good solution for the master problem. According to Ovacik (1997), the interactions between sub-problems come from two sources: job arrival times at a given sub-problem from the upstream shops, and job delivery time required for each job to complete its processing after leaving the current sub-problem. These are determined by the scheduling decisions taken at all previously scheduled sub-problems. Once a sub-problem is scheduled, constraints are imposed on the remaining sub-problems. The time by which a job is ready for processing at a particular sub-problem depends on the job precedence relationship at previous sub-problems. This defines the job arrival times for the current sub-problem ($r_j$). On the other hand, it is also important to estimate for how long the job will be stay in the shop after it leaves current sub-problem, which is denoted as delivery time ($q_j$). These arrival times and delivery times form the input to schedule the current sub-problem.

However, since we do not have the schedule for each sub-problem at the beginning, the arrival times and delivery times cannot be determined. To resolve this issue, unscheduled sub-problems are assumed to have infinite capacity so that the precedence relationships between jobs are ignored temporarily. Based on this assumption, a lower bound of the arrival times ($r_j$) can be obtained by summating the processing times of the job in upstream shops (5-1), and delivery times ($q_j$) can be calculated by adding the processing times in downstream shops (5-2).

$$r_j = \sum_{i \in U} p_{ji} \quad (5\text{-}1)$$

$$q_j = \sum_{i \in D} p_{ji} \quad (5\text{-}2)$$

Where

*U* is the set of stages prior to the current sub-problem.

*D* is the set of stages in the downstream of the current sub-problem.

Whenever a sub-problem is scheduled, the makespan of the whole problem will be prolonged resulting from the newly added job precedence relationship. Meanwhile, new constraints will impose on the arrival times and delivery times at the remaining sub-problems. The arrival time is set as the job completion time in the previous stage (5-3), and the delivery time can be obtained by calculating the shortest path at downstream stages (5-4).

$$r_j = w_{j,i-1} \quad (5\text{-}3)$$

$$q_j = STP_{j,D} \quad (5\text{-}4)$$

Where

$w_{j,i-1}$ is the completion time of job j at stage i-1.

$STP_{j,D}$ is the shortest path for job j at downstream stages. It can be obtained by calculating the earliest job completion time in the reversed downstream stages with predefined job precedence relationship.

The sub-problem for batching stage is modeled as parallel batch processing machines scheduling problem with non-identical arrival times, delivery times and arbitrary job sizes, and the objective is to minimize the time by which all jobs are delivered. The sub-problem can be presented as $P_m|r_j,\ q_j, s_j, batch|C_{max}$ , with $r_j$ and $q_j$ calculated by using equations 5-1, 5-2, 5-3 and 5-4. When batch processing machines are located at the last stage, $q_j$ equals 0. On the other hand, when the batch processing machines are at the first stage, $r_j$ equals to 0. Both scenarios are special cases of $P_m|r_j, q_j, s_j\ batch|C_{max}$ . Therefore, the same sub-problem formulation can be used for scheduling the batching stage no matter where it locates.

Table 5-1 Sub-problem Formulation

<table>
<tr><th>Sub-problem</th><th>Position</th><th>Formulation</th><th>Arrival Time ($r_j$)</th><th>Delivery Time ($q_j$)</th></tr>
<tr><td rowspan="3">PB</td><td>Front</td><td rowspan="3">$P_m|r_j, q_j, s_j, batch|C_{max}$</td><td>0</td><td rowspan="2">$q_j = \sum_{i \in D} p_{ji}$<br>or<br>$q_j = STP_{j,D}$</td></tr>
<tr><td>Interior</td><td rowspan="3">$r_j = \sum_{i \in U} p_{ji}$<br>or<br>$r_j = w_{j,i-1}$</td></tr>
<tr><td>Rear</td><td rowspan="2">0</td></tr>
<tr><td>DHFS</td><td>Downstream</td><td rowspan="2">$FF_m|r_j,\ q_j|C_{max}.$</td></tr>
<tr><td>UHFS</td><td>Upstream</td><td>0</td><td>$q_j = \sum_{i \in D} p_{ji}$<br>or<br>$q_j = STP_{j,D}$</td></tr>
</table>

The upstream HFS and downstream HFS can be both modeled as hybrid flow shop scheduling problem with arrival times, delivery times and the objective to minimize

the time by which all jobs are delivered, or $FF_m|r_j,\ q_j|C_{max}$. For upstream HFS, arrival time $r_j$ equals zero, and the sub-problem can be simplified as $FF_m|\ q_j|C_{max}$. The delivery time $q_j$ is determined by the schedule at downstream stages. For downstream HFS, $q_j$ equals to zero, and the sub-problem can be simplified as $FF_m|r_j,|C_{max}$. The arrival time $r_j$ depends on the schedule at upsteam stages.

An overview of sub-problem formulation is given in Table 5-1.

## 5.3 Sub-problem Prioritization

According to the theory of constraint, the sub-problem with more critical resource constraint has the higher priority to be scheduled. Therefore, sub-problems are scheduled in the order of non-decreasing criticality of resource constraints. In this research, *Cmax* is used to measure the criticality of resource constraints. The sub-problem with the largest *Cmax* has the highest scheduling priority.

## 5.4 Sub-problem Solving

Selection of the sub-problem solution approach depends on a tradeoff between time and quality. In original shifting bottleneck procedures, branch-and-bound is applied to find the optimal solution for single machine scheduling problem, but it is computationally expensive, and the shifting bottleneck procedure built on branch and bound can only solve up to ten-job and ten-stage problem within a minutes. Since the problem under study contains batch processing machines, it is even more complicated, hence an exact method like branch and bound will require even more time to solve the

sub-problem. To reduce the computational cost, heuristic methods will be applied to solve sub-problems efficiently.

### 5.4.1 *Sub-problem 1: Scheduling Parallel Batch Processing Machines with Arrival Times, Delivery Times and Non-identical Job Sizes*

#### 5.4.1.1 Problem Description

This sub-problem can be described as follows: a set $J$ of jobs and a set $M$ of identical batch processing machines with capacity constraint $C$ are given. Each job $j \in J$ is described by ($p_j$, $r_j$, $q_j$, $s_j$) representing its processing time, arrival time, delivery time and size, respectively. The objective is to find a set $B$ of batches and to schedule these batches such that the time by which all jobs are delivered is minimized. The scheduling problem is denoted as $P_m|r_j, q_j, s_j, batch|C_{max}$, and is known to be strongly NP-hard.

#### 5.4.1.2 Proposed Solution Approach

To the best of author's knowledge, the literature on heuristics for the $P_m|r_j, q_j, s_j, batch|C_{max}$ can not be found. Two most close researches on a reduced problem, $P_m|r_j, s_j, batch|C_{max}$, are conducted by Chung (2008) and by Vélez-Gallego (2009). Both researches proposed the heuristics based on *batching first, sequencing second* strategy. The heuristics solve the problem in two phases: first a batching procedure is applied to form the batches, and then scheduling rules (earliest ready time, or longest earliest completion time) are used to schedule the batches formed. However this two-phase procedure has a drawback which might hurt the solution quality: if a job is ready before a machine is available, the ready time of this job should no longer be a valid

constraint. The two-phase procedure does not take into account this fact, so that might lead to a bad solution by considering unnecessary ready time constraints. Another reduced problem, $P_m|r_j,\ q_j|C_{max}$, is studied by Gusfield (1984), Carlier (1987; 1998) and Gharbi (2007). They investigated the so-called Jackson's algorithm to minimize the time by which all jobs are delivered. Jackson's algorithm is a simple list scheduling algorithm based on a dispatching rule which schedules, on the earliest available machine, the available job with a largest delivery. Since Jackson's algorithm is originally developed for discrete machine scheduling problem, it does not deal with batching, therefore cannot be applied directly to solve our sub-problem $P_m|r_j,\ q_j, s_j, batch|C_{max}$.

In this research, a new heuristic called List-scheduling-with-delay (LSD) is developed to solve $P_m|r_j,\ q_j, s_j, batch|C_{max}$. LSD incorporates the merits of the list scheduling and the Delay Heuristic proposed by Lee and Uzsoy (1999). It allows for postponement in processing a batch in order to accommodate a job that is due to arrive soon and might be combined with the delayed batch as long as the machine capacity constraint is not violated. Instead of separating batching procedure from scheduling procedure, LSD integrates these two procedures. Batches are formed and scheduled concurrently. Whenever a new batch is formed and scheduled, machine available time will be updated. If a job is ready by the time when a machine is available, its ready time constraint will thus become invalid. All the unscheduled jobs which are ready to be processed will then form a set of job candidates to be assigned to that machine. By doing this, LSD avoids the unnecessary ready time constraints and thus improves the solution quality.

Let $t$ be the earliest time when at least one machine and one job are available, and a certain period of delay has passed. At time $t$, amongst ready jobs $Jt$, starting from the one with the longest summation of processing time and delivery time $(p_j + q_j)$, jobs are added one by one into current batch as long as the machine capacity is not exceeded. If all $q_j$ equal to zero, the dispatching rule is then reduced to LPT. The batch is then assigned to the first available machine $l$, and the machine available time is updated. Batching process is repeated until all jobs have been scheduled. The procedure of LSD is shown in Figure 5-4.

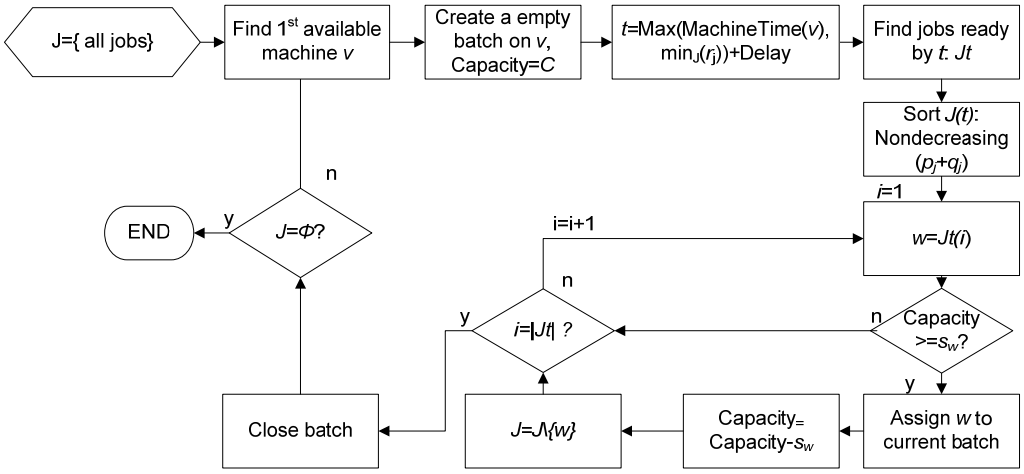



Figure 5-4 Procedure of LSD

The delay period is a parameter for the LSD heuristic, and the value is set by following equation:

$$delay_k = \frac{max_{j\in J}\{r_j\} - min_{j\in J}\{r_j\}}{n-1} \times k, \quad \forall k = 0,1,\dots n-1 \qquad 5\text{-}5$$

Where $n$ is the number of jobs.

For each problem instance, the LSD algorithm will be run $n$ times, each with a different *delay* value calculated from the equation 5-5. After $n$ iterations, the best result is reported. Taking advantage of the reversibility feature, the LSD algorithm can be also applied backwardly on the problem instance, with a reversed routing and time frame, and the job arrival times and delivery times switched. The better solution between the forward result and backward result is chosen as the final solution.

#### 5.4.1.3 A Numerical Example

A numerical example is given here to illustrate the algorithm. There are two batch processing machines in parallel, each with 10-unit capacity. The processing times, sizes, arrival times and delivery times of the jobs are shown in Table 5-2.

Table 5-2 Data of the Numerical Example

| Jobs | $p$ | $s$ | $r$ | $q$ | $p+q$ |
|---|---|---|---|---|---|
| Job 1 | 7 | 3 | 6 | 0 | 7 |
| Job 2 | 8 | 8 | 6 | 9 | 17 |
| Job 3 | 3 | 3 | 4 | 3 | 6 |
| Job 4 | 8 | 4 | 3 | 5 | 13 |
| Job 5 | 6 | 5 | 7 | 7 | 13 |
| Job 6 | 5 | 2 | 8 | 4 | 9 |

To simplify the process, assume *k=4* in this iteration.

First, we calculate the delay period:

$$delay = \frac{(\max_{j\in J} r_j) - (\min_{j\in J} r_j)}{n-1} \times 4 = \frac{8-3}{5} \times 4 = 4$$

The procedure of scheduling can be found in Table 5-3.

Table 5-3 Scheduling Procedure of Numerical Example

| $J$ | $t$ | $J_t$ | $w$ | Machine 1 | Machine 2 | Cmax' |
|---|---|---|---|---|---|---|
| 1,2,3,4,5,6 | 7 | 2, 4, 5, 1, 3 | 2 | (2) | | 23 |
| 1,3,4,5,6 | 7 | 4, 5, 1, 3 | 4 | | (4) | 16 |
| 1,3,5,6 | | 5, 1, 3 | 5 | | (4,5) | 22 |
| 1,3,6 | 18 | 6,1,3 | 6 | (6) | | 23 |
| 1,3 | | 1,3 | 1 | (6,1) | | 25 |
| 1 | | 3 | 3 | (6,1,3) | | 25 |

At the beginning of the scheduling, a decision point in time $t$ is calculated by $\max(0, min_{j\in J} r_j) + delay = 3 + 4 = 7$. All jobs ready by time $t$ are added to a candidate set $J_t$, and ranked in non-deceasing order of $p_j + q_j$. The job sequence in $J_t$ is (2, 4, 5, 1, 3). The first job, job 2, is assigned to a new batch, and removed from both $J$ and $J_t$. Since $S_2$=8, job 2 cannot accommodate any other job in the same batch, this batch is closed and be processed on machine1. $Cmax' = r_2 + p_2 + q_2 = 8 + 6 + 9 = 23$. This cycle repeats until all jobs are scheduled. The $Cmax'$ obtained is 25.

The final schedule is obtained as below:

Machine 1: (2) $\rightarrow$ (6, 1, 3)

Machine 2: (4, 5)

Cmax=25.

The solution is proved to be optimal by CPLEX.

5.4.1.4 Evaluation of Solution Quality

An experiment is performed to evaluate the solution quality of LSD against other two heuristics from literatures, the JobSimilarity (JS) proposed by Vélez-Gallego (2009)

and the ModifiedDelay (MD) proposed by Chung (2008). All three heuristics are tested on the same set of random instances generated in the same way as Vélez-Gallego did in his research. A problem instance is defined by a set of $n$ jobs and a set of $m$ machines. Each job $j$ is described by the triplet $\{p_j, r_j, s_j\}$, and each machine is described by its capacity $S$. The processing times, ready times and job sizes are sampled from a discrete uniform (DU) random variable so that $p_j \sim DU[1, MaxP]$, $r_j \sim DU[1, \rho Z]$ and $s_j \sim DU[1, MaxS]$, where $Z = \sum_{j \in J} p_j$ and $0 \leq \rho \leq 1$.The factors are given in Table 5-4.

Table 5-4 Experimental Factors

| Factors | Level | Factors | Level |
|---|---|---|---|
| $n$ | 10, 20, 50, 100, 200 | $\rho$ | 5%, 10%, 50% |
| $m$ | 3, 5 | *MaxP* | 10, 30 |
| *MaxS* | 5, 20 | *Machine Capacity* | 20 |

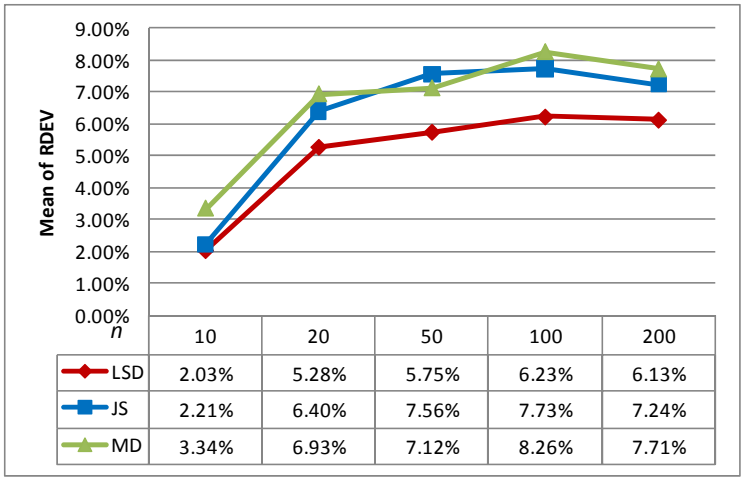



Figure 5-5 Solution Quality of LSD, JobSimilarity and ModifiedDelay

Experiments result reveals that LSD outperforms both JobSimilarity and ModifiedDelay, which, to the best of our knowledge, are the only two heuristic approaches available in the literature for the parallel batching machines scheduling problem with job ready times and non-identical job sizes.

### 5.4.2 *Sub-problem 2: Scheduling the Hybrid Flow Shop with Arrival Times and Delivery Times*

#### 5.4.2.1 Problem Description

This sub-problem can be described as below: There are multiple stages of machines in series, each stage comprises of one or several identical machines in parallel. There are a set of jobs $J$ to be processed following the same routing. Each job $j \in J$ is described by ( $p_j$, $r_j$, $q_j$ ) representing its processing time, arrival time and delivery time, respectively. The objective is to find a schedule that minimizes the time by which all jobs are delivered. This sub-problem can be denoted as $FF_m|r_j,\ q_j|C_{max}$, and is NP hard in strong sense.

#### 5.4.2.2 Proposed Solution Approach

##### *5.4.2.2.1 Overview*

A bottleneck-based heuristic, named Bottleneck Approach based on Jackson's Heuristic (BA-Jackson), is proposed to solve this sub-problem. BA-Jackson incorporates the merits from bottleneck-based heuristics and a modified Jackson's heuristic. In the BA-Jackson heuristic, a bottleneck stage is first identified and scheduled by applying the

Modified Jackson's Heuristic, and then upstream and downstream stages are scheduled sequentially using the same heuristic. The major procedure is show in Figure 5-6.

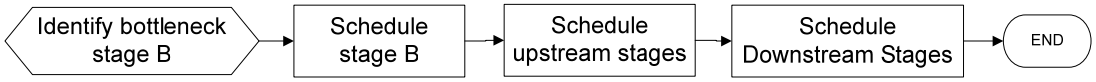



Figure 5-6 Procedure of BA-Jackson

*5.4.2.2.2 Proposed Modified Jackson's Heuristic*

The well-known Jackson's Heuristic is originally developed for parallel machines scheduling problem with arrival delivery times to minimize the time by which all jobs can be delivered. The heuristic can be states as follows: on the earliest available machine, schedule the available job with the largest delivery time. In this research, the Jackson's Heuristic is modified by applying a delay strategy to improve the solution quality. With the modification, it is possible that a job with late arrival time but large delivery time be processed before a job with early arrival time but less delivery time. The Modified Jackson's Heuristic is described as below: Let $t$ be the earliest time when at least one machine and one job are available, and a certain period of delay has passed. At time $t$, amongst the ready jobs, assign a job with maximum delivery time $q_j$ to the first available machine, breaking tie with LPT rule. Same as LSD, the value of delay period is determined by equation 5-5. For each problem instance, the Modified Jackson's Heuristic will be run $n$ times, each with a different *delay* value calculated from equation 5-5. After $n$ iterations, the best result is reported as the final solution. The procedure of the Modified Jackson's Heuristic is given in Figure 5-7.

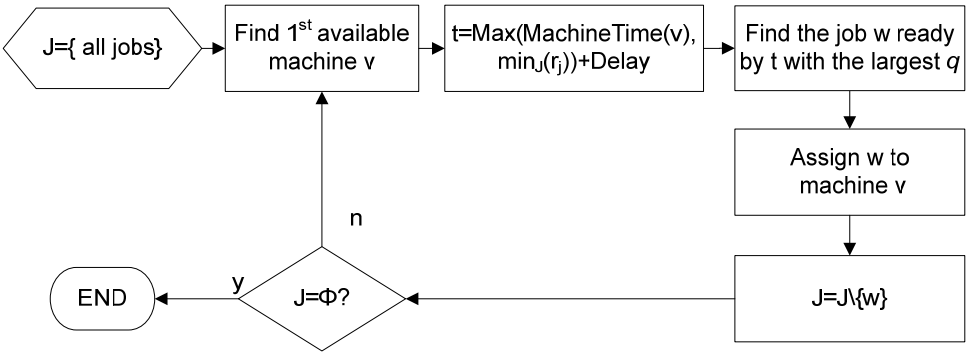



Figure 5-7 Modified Jackson's Heuristic with Given Delay Parameter

*5.4.2.2.3 The Proposed BA-Jackson*

The procedure of the BA-Jackson heuristic is given as follows:

a. Identify the bottleneck stage

$WR_i \leftarrow \dfrac{\sum_{j \in J} p_{ji}}{m_i}$ //For each stage i, compute the workload ratio.

$B \leftarrow \arg\max_{i \in I} WR_i$ //Select the stage $B$ with maximum workload.

b. Schedule stage *B*

Apply Modified Jackson's Heuristic to schedule the bottleneck stage *B*, which is formulated as $P_m|r_{jB}, d_{jB}|C_{max}$. Job arrival times $r_{jB}$, and job delivery times $d_{jB}$ are calculated as follows:

$$r_{jB} = \begin{cases} r_j, & B = 1 \\ r_j + \sum_{s=1}^{B-1} p_{js}, & B > 1 \end{cases} \quad \text{5-6}$$

$$q_{jB} = \begin{cases} q_j + \sum_{s=B+1}^{|I|} p_{js}, & \forall\, 1 < B < |I| \\ q_j, & B = |I| \end{cases} \quad \text{5-7}$$

Once schedule is decided and the job delivered time ($dt_j$) is calculated for each job, the added value of $dt_j$ due to new imposed job precedence relationship at stage *B* can be calculated by following equation:

$$Add_j = dt_j - \sum_{i \in I} p_{ji} \ , \qquad \forall\, 1 < \ B < |I| \tag{5-8}$$

c. Schedule upstream stage

Starting from stage 1, apply Modified Jackson's Heuristic to schedule each stage sequentially. Job arrival times $r_{ji}$, and job delivery times $d_{ji}$ are calculated as follows:

$$r_{ji} = \begin{cases} r_j, & i = 1 \\ ct_{j,i-1}, & \forall 1 < i < B \end{cases} \tag{5-9}$$

$$q_{ji} = Add_j + \sum_{s=i+1}^{|I|} p_{js} \ , \qquad 1 < \ i < B \tag{5-10}$$

d. Schedule downstream stage

Starting from stage *B+1*, apply Modified Jackson's Heuristic to schedule each stage sequentially. Job arrival times $r_{ji}$, and job delivery times $d_{ji}$ are calculated as follows:

$$r_{ji} = ct_{j,i-1}, \qquad \forall B < i \leq |I| \tag{5-11}$$

$$q_{ji} = \begin{cases} \sum_{s=i+1}^{|I|} p_{js} \ , & \forall\, B < \ i < |I| \\ q_j, & i = |I| \end{cases} \tag{5-12}$$

## 5.5 Re-scheduling

Re-scheduling, also called re-optimization, is an essential element of decomposition algorithms and proved to be very effective in improving the solution quality. It is sometimes also referred to as the control structure. Whenever a new sub-problem has been scheduled, all the sub-problems that have been scheduled previously need to be rescheduled in the non-increasing order of their criticality of resource constraints. More specifically, for each sub-problem previously scheduled, the schedule obtained before is discarded and treated as without job precedence relationship. The same method for solving the sub-problem as described before is then applied to obtain a new schedule. This may lead to a better schedule because more information has become available with regard to the other sub-problem at this moment.

## 5.6 Algorithm of the BFD Heuristic

The BFD heuristic includes a main procedure and a sub-procedure. The sub-procedure called Bottleneck Identification and Scheduling (BIS) is called by the main BFD procedure iteratively to identify and schedule the bottleneck sub-problem. Both procedures are described in this section.

### 5.6.1 Main Procedure

The flow chart of the main procedure is given in Figure 5-8.

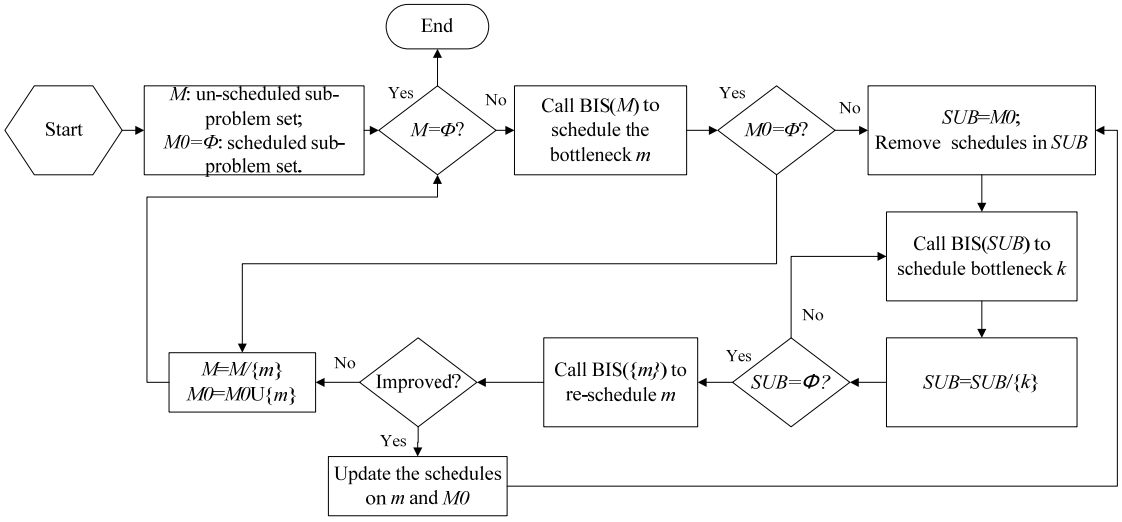



Figure 5-8 Main Procedure of BFD

The procedure is stated as follows:

Step 1: Initialization

If $B = 1$

$M \leftarrow \{PB, DHFS\}$.

Else if $B = |I|$

$M \leftarrow \{UHFS, PB\}$.

Else

$M \leftarrow \{UHFS, PB, DHFS\}$.

End if

$M0 \leftarrow \Phi$

Step 2: Identify and schedule the bottleneck sub-problem

Call BIS($M$) to identify and schedule the bottleneck sub-problem $m \in M$

Step 3: Re-scheduling

1) $SUB \leftarrow M0$
2) Remove all the job precedence relationship in $SUB$

3) Call BIS(*SUB*) to identify and schedule the bottleneck sub-problem $k \in SUB$

4) $SUB \leftarrow SUB/\{k\}$

5) If $SUB \neq \Phi$

go to 3)

Else

Call BIS($\{m\}$) to re-schedule *m*

If the scheduling result is improved

Update schedules on *m* and *M0*

Go to 1)

Else

$M0 \leftarrow M0 \cup \{m\}$

$M \leftarrow M/\{m\}$

Go to step 4

End if

Step 4: If $M = \emptyset$, go to step 2, otherwise, stop

### 5.6.2 Sub-procedure: BIS

The flow chart of the sub-procedure BIS is given in Figure 5-9.

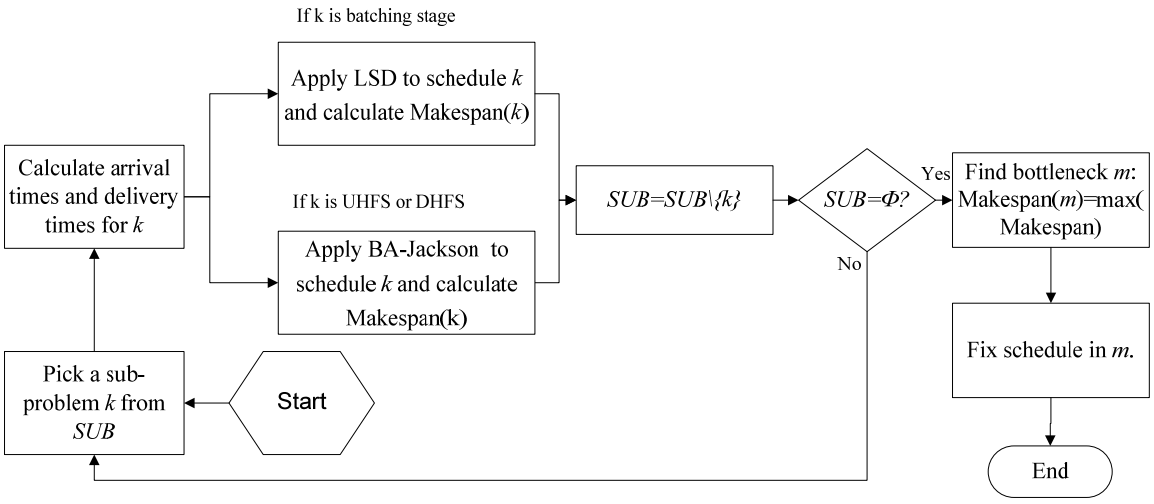



Figure 5-9 Sub-procedure: Bottleneck Identification and Scheduling

The procedure is stated as follows:

Step 1: Randomly pick a sub-problem $k$ from $SUB$

Step 2: Calculate arrival times and delivery times for $k$

$$r_{jk} = \begin{cases} 0, & if\ k = PB\ front\ or\ UHFS \\ ct_{j,k-1}, & if\ k = PB\ interior, PB\ rear\ or\ DHFS \end{cases}$$

$$q_{jk} = \begin{cases} ct'_{j,k+1}, & if\ k = PB\ front, PB\ interior\ or\ UHFS \\ 0, & if\ k = PB\ rear\ or\ DHFS \end{cases}$$

Where

$ct_{j,k-1}$ is the job completion time of the sub-problem before $k$.

$ct'_{j,k+1}$ is the job completion time of the sub-problem after $k$ based on non-delay reversed schedule.

Step 3: Formulate and solve the sub-problem $k$:

If $k$ =PB,

Apply LSD to schedule $k$.

Calculate Cmax($k$).

Else

Apply BA-Jackson to schedule $k$.

Calculate Cmax($k$).

End

Step 4: Set $SUB \leftarrow SUB/\{k\}$. If $SUB \neq \Phi$, go to step 1, otherwise go to step 5.

Step 5: Find the sub-problem $m$ with maximum value of Cmax. $m$ is considered as the bottleneck.

Step 6: Fix the job precedence relationship on $m$, and remove the job precedence relationship from all other sub-problems in the original set *SUB*.

## 5.7 A Numeric Example

A simple example is used to illustrate the procedure of BFD described in 5.6.1. This example includes 6 jobs and 3 stages. The first stage has two batch processing machines in parallel, each with 10 units of capacity. The second stage has only one single discrete machine, and the last stage has two discrete machines in parallel. The processing times and sizes of the jobs are shown in Table 5-5.

To simplify the calculation, CPLEX is used to solve all the sub-problems.

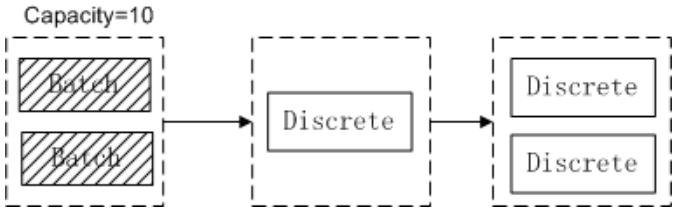



Figure 5-10 Shop Configuration of the Example

Table 5-5 Data of the Numerical Example

| Jobs | $s$ | $p_1$ | $p_2$ | $p_3$ |
|---|---|---|---|---|
| Job 1 | 4 | 47 | 5 | 19 |
| Job 2 | 4 | 77 | 10 | 17 |

| | | | | |
|---|---|---|---|---|
| Job 3 | 3 | 21 | 7 | 9 |
| Job 4 | 6 | 94 | 4 | 18 |
| Job 5 | 4 | 58 | 10 | 8 |
| Job 6 | 5 | 38 | 6 | 3 |

Following the BFD main procedure, the solution is derived step by step as below:

Step 1: Initialization

Divide the problem into parallel batch processing machines (*PB*), and downstream hybrid flow shop (*DHFS*): *M*={*PB, DHFS*}.

$M0 = \emptyset$.

Step 2: Identify and schedule the bottleneck sub-problem

Use AMPL/CPLEX to identify and schedule the bottleneck. Sub-problem formulations and schedule obtained are shown in Table 5-6 and Table 5-7.

Table 5-6 Sub-problem Formations and Scheduling

| Sub-problem | Formulation | Arrival Time | Delivery Time | Cmax |
|---|---|---|---|---|
| *PB* | $P_m\|q_j, s_j, batch\|C_{max}$ | 0 | $q_j = \sum_{i\in D} p_{ji}$ 24, 27,16,22,18,9 | 121 |
| *DHFS* | $FF_m\|r_j\|C_{max}$ | $r_j = p_{j1}$ 47,77,21,94,58, 38 | 0 | 116 |
| Max (Cmax)=121, therefore bottleneck is *PB* | | | | |

Table 5-7 Job Sequences on Parallel Batch Processing Machines

| Machine | Job Sequence |
|---|---|
| Machine 1 | $(1, 5) \rightarrow (3,6)$ |
| Machine 2 | (2, 4) |

Step 3: Re-scheduling

Since there is only one sub-problem scheduled so far, re-scheduling is not required.

*M*={ *DHFS*}

*M0*={ *PB*}

Since there is still a sub-problem in the unscheduled problem set *M*, Step 2 need to be repeated to solve the remaining sub-problem.

Step 2: Use CPLEX to schedule the only sub-problem, *DHFS*, in *M*. The formulation and scheduling are shown in Table 5-8 and Table 5-9.

Table 5-8 Sub-problem Formulation and Scheduling

| Subsystem | Formulation | Arrival Time | Delivery Time | Cmax |
|---|---|---|---|---|
| *DHFS* | $FF_m \mid r_j \mid C_{max}$ | $r_j = ct_{1j}$<br>58, 94, 96, 94, 58, 96 | 0 | 128 |
| Since *DHFS* is the only un-scheduled sub-problem, it is also the bottleneck. | | | | |

Table 5-9 Job Sequences on the Down Stream Hybrid Flow Shop

| Stage | Machine | Job Sequence |
|---|---|---|
| 2nd stage | Machine 1 | $5 \rightarrow 1 \rightarrow 4 \rightarrow 2 \rightarrow 3 \rightarrow 6$ |

| | | |
|---|---|---|
| 3rd stage | Machine 1 | $5 \rightarrow 1 \rightarrow 4 \rightarrow 3 \rightarrow 6$ |
| | Machine 2 | 2 |

Step 3: Re-scheduling

$SUB = M0 = \{ PB \}$.

Remove all the job precedence relationship in *SUB*.

Use AMPL/CPLEX to schedule *PB*. The results are given in Table 5-10 and Table 5-11.

Table 5-10 Sub-problem Formations and Scheduling

| Sub-problem | Formulation | Arrival Time | Delivery Time | Cmax |
|---|---|---|---|---|
| *PB* | $P_m \mid q_j, s_j, batch \mid C_{max}$ | 0 | $q_i = ct'_D$<br>54, 29, 19, 34, 67, 9 | 128 |
| Since *PB* is the only un-scheduled sub-problem, it is also the bottleneck | | | | |

Table 5-11 Job Sequences on Parallel Batch Processing Machines

| Machine | Job Sequence |
|---|---|
| Machine 1 | $(1, 5) \rightarrow (3) \rightarrow (6)$ |
| Machine 2 | (2, 4) |

Use AMPL/CPLEX to re-schedule *DHFS*. The results are given in Table 5-12 and Table 5-14.

Table 5-12 Sub-problem Formations and Scheduling

| Sub-problem | Formulation | Ready Times | Cmax |
|---|---|---|---|
| *DHFS* | $FF \mid r_j \mid C_{\max}$ | 58, 94, 79, 94, 58, 117 | 126 |

| |
|---|
| 126<128, Cmax is improved. |

Table 5-13 Job Sequence on the Down Stream Hybrid Flow Shop

| Stage | Machine | Job Sequence |
|---|---|---|
| 2nd stage | Machine 1 | $1 \rightarrow 5 \rightarrow 3 \rightarrow 2 \rightarrow 4 \rightarrow 6$ |
| 3rd stage | Machine 1 | $1 \rightarrow 5 \rightarrow 2 \rightarrow 6$ |
| | Machine 2 | $3 \rightarrow 4$ |

Since the scheduling result is improved, re-scheduling need to be repeated. The second iteration of re-scheduling shows no improvement, therefore re-scheduling stops after two iterations of re-scheduling. The detail of the second re-scheduling is skipped here.

$M0 = M0 \cup \{DHFS\} = \{PB, DHFS\}$.

$M = M/\{DHFS\} = \emptyset$.

Step 4: Since all sub-problems have been scheduled, BFD stops. The final schedule is shown in Table 5-14 with Cmax=126.

Table 5-14 Final Job Sequence

| Stage | Machine | Job Sequence |
|---|---|---|
| 1st stage | Machine 1 | $(1, 5) \rightarrow (3) \rightarrow (6)$ |
| | Machine 2 | (2, 4) |
| 2nd stage | Machine 1 | $1 \rightarrow 5 \rightarrow 3 \rightarrow 2 \rightarrow 4 \rightarrow 6$ |
| 3rd stage | Machine 1 | $1 \rightarrow 5 \rightarrow 2 \rightarrow 6$ |
| | Machine 2 | $3 \rightarrow 4$ |

Using CPLEX to solve this instance, the Cmax is also 126. However, if sub-problem is not solved by AMPL/CPLEX but by the sub-problem heuristic LSD and BA-Jackson, the Cmax obtained is 128. This is because the solution of sub-problem produced by heuristics is not optimal, which affects the solution quality for the whole problem.

## 5.8 Computational Experiment

### 5.8.1 Experiment Design

The BFD heuristic is first compared with two reduced versions of BFD to examine the effectiveness of re-scheduling process, and then further compared against several commonly used heuristics, including a Randomized Greedy Heuristic and five well-known dispatching rules, to evaluate its solution quality. All comparisons conducted are listed in Figure 5-11.

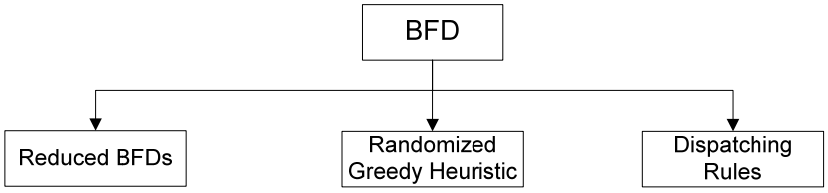



Figure 5-11 List of Comparisons

#### 5.8.1.1 Reduced BFD Heuristics

To verify the significance of re-scheduling procedure, two reduced versions of BFD heuristic, BFD1 and BFD2 (Table 5-15), are used to provide comparison.

In original BFD, whenever a sub-problem is scheduled, the previously scheduled sub-problems will be rescheduled based on the job sequences on the last scheduled sub-

problem. This re-scheduling procedure is referred to as full re-scheduling and is computationally intensive. To reduce computational cost, BFD1 completely skips the re-scheduling procedure, while BFD2 considers an alternative re-scheduling procedure: re-scheduling is performed only after all sub-problems have been introduced into the schedule.

Table 5-15 BFD and Reduced BFDs

| Algorithm | Re-scheduling |
|---|---|
| BFD | Full |
| BFD1 | None |
| BFD2 | Last Iteration Only |

All the algorithms are tested on the same set of problem instances, and results will be analyzed to evaluate the importance of the re-scheduling procedure.

#### 5.8.1.2 Randomized Greedy Heuristic

A greedy heuristic is a heuristic that follows the problem solving approach of making the locally optimal choice at each stage with the hope of finding the global optimum (Black 2005). The greedy randomized solutions are generated by adding elements to the problem's solution set from a list of elements ranked by a greedy function. Ranked candidate elements are often placed in a restricted candidate list (RCL), and chosen at random when building up the solution. This kind of greedy randomized construction method is also known as randomized greedy heuristic (RGH) or a semi-greedy heuristic, first described by Hart (1987).

In the proposed RGH, the job candidates ready for processing are ranked according to the Longest Remaining Processing Time First rule (LRPT). Instead of

selecting the first job in the list, the job is selected randomly from the top $k$ elements in the list, where $k$ is the size of the RCL and predefined as 3 in this dissertation. The selected element is then assigned to a batch (in batch processing stage) or a machine (in discrete stage).

The procedure of the proposed RGH is as follows:

Starting from the first stage, schedule jobs at each stage using following algorithm:
Let current stage number be $i$
Let $J$ be the set of unscheduled jobs
Let $M$ be the set of machines in parallel
Let $r_{ji}$ be the ready time of job $j$ at stage $i$. $r_{ji}$ is set as 0 when $i$=1, otherweise set $r_{ji}$ as the completion time of job $j$ in stage $i$-$1$.
$MachineAvailTime_{m\in M} \leftarrow 0$
If stage $i$ is the batch processing stage
$B \leftarrow \Phi$
$b \leftarrow 0$
While $J \neq \Phi$ do
$B \leftarrow B+1$
$B_b \leftarrow \Phi$
*Capacity*=$S$
$l \leftarrow argmin_{m\in M} MachineAvailTime_m$
$t1 \leftarrow MachineAvailTime_l$
$t2 \leftarrow \min_{j\in J} r_j$
$t \leftarrow \max(t1, t2)$
$J_t \leftarrow \{j \in J \mid r_j \le t\}$
Sort $J_t$ in LRPT, breaking tie with LPT
$MachineAsign_b \leftarrow l$
For $i$=1 to $|J_t|$
$RCL \leftarrow J_t(1:k)$
Randomly choose a job $w$ from $RCL$

if $Capacity - s_w \geq 0$

$B_b \leftarrow B_b \cup \{w\}$

$J \leftarrow J \setminus \{w\}$

$J_t \leftarrow J_t \setminus \{w\}$

$Capacity \leftarrow Capacity - s_w$

End if

End for

End while

Else if stage *i* is a discrete stage

While $J \neq \Phi$ do

$l \leftarrow argmin_{m \in M} MachineAvailTime_m$

$t1 \leftarrow MachineAvailTime_l$

$t2 \leftarrow \min_{j \in J} r_j$

$t \leftarrow \max(t1, t2)$

$J_t \leftarrow \{j \in J \mid r_j \leq t\}$

Sort $J_t$ in LRPT, breaking tie with LPT

$RCL \leftarrow J_t(1:k)$

Randomly select a job *w* from *RCL*

$MachineAsign_b \leftarrow l$

$J \leftarrow J \setminus \{w\}$

$J_t \leftarrow J_t \setminus \{w\}$

$st_w \leftarrow max(MachineAvailTime_l, r_w)$

$ct_w \leftarrow st_w + p_w$

$MachineAvailTime_l \leftarrow ct_w$

End while

End

#### 5.8.1.3 List Scheduling With Dispatching Rules

Five commonly used dispatching rules combined with list scheduling are adopted to provide comparisons. By applying dispatching rule, every time when a machine becomes available, one job among the set of unscheduled jobs is selected as the next one on the machine according to a certain priority index. Dispatching rules used in the experiment are listed as below:

1. List Scheduling with Largest Delivery Time (LS-LDT): Select the available job with largest delivery time.
2. List Scheduling with Largest Remaining Processing Time (LS-LRPT): Select the available job with largest amount of remaining processing time.
3. List Scheduling with Largest Processing Time (LS-LPT): Select the available job with largest processing time.
4. List Scheduling with Earliest Ready Time (LS-ERT): Select the available job that has been ready for the longest time.
5. List Scheduling with Shortest Processing Time (LS-SPT): Select the available job with the shortest processing time.

For all above dispatching rules, machine assignment is made implicitly according to the available timing of machines, and batch forming and scheduling is done by following algorithm:

1. Find out the first available machine, and open a new batch on that machine.
2. Find out all the unscheduled jobs $\boldsymbol{J_t}$ which are ready to be process at current moment $t$, and change the ready times of all jobs in $\boldsymbol{J_t}$ to $t$.
3. Sort jobs in ERT, breaking ties using the predefined dispatching rule.

4. Assign the jobs one by one from the top of the list to current batch if the batch size doesn't exceed the machine capacity.
5. When no more jobs can fit in current batch, close the batch.
6. Go to step 1, and repeat the above procedure until all jobs are scheduled.

The result obtained by the BFD heuristic is compared with the result by each dispatching rule, as well as the best result among them, which is referred to as BEST-DSPT. BEST-DSPT is obtained by running all dispatching rules on each problem instance and selecting the best solution obtained. The computational time of this procedure is given by the sum of the times for all dispatching rules.

### 5.8.2 Data Generation

Experiment data is randomly generated in a manner similar to the methods by Chen (2009) and Kim (2009), with additional considerations of non-identical job sizes and batch processing machines. Table 5-16 summarizes the experimental factors: number of jobs, number of stages, size of jobs, position of batching stage, position of bottleneck stage and work load difference between bottleneck and non-bottleneck stages. Number of jobs has six levels: 4, 6, 10, 20, 50 and 100. Number of stages has three levels: 3, 5 and 7 (low, medium and high), where each batching stage has 2 to 4 machines in parallel, and each discrete stages has 1 to 4 machines in parallel. For small problem instances with less than or equal to 10 jobs, the number of machines is 2 at the batching stage, and 1 to 3 at each discrete stage. The capacity of batch processing machines is assumed to be 10 in small instances, and 20 in median and large instances, and the size of job is generated from discrete uniform distribution of [1, 5] (small), [1, 9] (mixed) and [3, 7] (big). The

position of batching stage has three levels: front, interior and rear (the first stage, any interior stage and the last stage). The exact position of the interior stage is randomly selected from discrete uniform distribution of [2, $\boldsymbol{v}-1$], with $v$ denoting number of stages. The position of bottleneck stage has the same levels as the position of batching stage, and is generated in the same way. The workload difference between the bottleneck stage and the highest work load of non-bottleneck stage has also three levels: the low one is randomly generated from uniform distribution of [1, 1.2], the median one is generated from [1.5, 1.9], and the high one is generated from [2.0, 3.0].

Table 5-16 Experiment Factors

| Experiment Factors | Level | |
|---|---|---|
| Number of Jobs ($n$) | 4, 6, 10, 20, 50, 100 | |
| Number of Stages ($v$) | Low | 3 |
| | Median | 5 |
| | High | 7 |
| Size of Jobs ($s$) | Small | U[1, 5] |
| | Mix | U[1, 9] |
| | Big | U[3, 7]) |
| Position of Batching Stage ($b$) | Front | 1 |
| | Interior | U[2, $\boldsymbol{v}-1$] |
| | Rear | $\boldsymbol{v}$ |
| Position of Bottleneck Stage ($q$) | Front | 1 |
| | Interior | U[2, $\boldsymbol{v}-1$] |
| | Rear | $\boldsymbol{v}$ |
| Workload Difference ($w$) | Low | U[1.0, 1.2] |
| | Median | U[1.5, 1.9] |
| | High | U[2.0, 3.0] |

The workload of a specified bottleneck stage is implemented as follows: (1) randomly generate the processing times for every job on each stage from discrete uniform

distribution of [1, 50]; (2) calculate workload $R_i$ for each non-bottleneck stage: $R_i = (\sum_j p_{ji})/m_i$ for every discrete stage, and $R_i = (\sum_j p_{ji} \times \bar{s})/m_i c$ for the batching stage, with $\bar{s}$ denoting average job size. The highest workload value $R_{\max}$ is set as the benchmark; (3) apply the same formula to calculate the workload for the bottleneck stage; (4) with a specified workload difference value ($w$), modify the processing time of job $j$ at bottleneck stage $b$: $p_{jb} = \frac{(old\_p_{jb}) \times R_{\max} \times w}{old\_R_b}$. This procedure will guarantee that $\frac{R_b}{R_{\max}}$ equals the specified $w$.

With six factors considered, there are a total of 1458 production scenarios. 1 problem instance for each scenario, there are 1458 problem instances in total.

### 5.8.3 Experiment Measurement

The relative deviation (RDEV) is used as the measurement to evaluate the solution quality of different algorithms. The RDEV is defined as:

$$RDEV = \frac{C\max_b - C\max_a}{C\max_a} \times 100\% \qquad 5\text{-}13$$

$C\max_b$ is the solution value obtained by method b. $C\max_a$ is the benchmark solution value. When the comparison is between heuristic and lower bound, $C\max_a$ is replaced by the lower bound value. If the comparison is made between heuristic and the optimal solution, $C\max_a$ will be the optimal solution. For problem instances which can be solved by AMPL/CPLEX to the optimality, the optimal solution is used as the

benchmark. For those instances whose optimal solution is unknown, lower bound is calculated to provide the benchmark.

### 5.8.4 Experiment Result

#### 5.8.4.1 Analysis of Variance

The analysis of variance (ANOVA) (Table 5-17) shows that three factors have statistically significant impact on BFD's solution quality at a 5% significance level. These factors include number of job ($n$), batching position ($b$) and bottleneck position ($q$). While the remaining three factors, number of stages ($v$), job size ($s$) and workload difference ($w$), do not show statistically significant impact. A main effects plot for each factor tested is presented in Figure 5-12.

Table 5-17 ANOVA Table for RDEV

| Factors | P Value |
|---|---|
| $n$ | 0.000 |
| $v$ | 0.697 |
| $s$ | 0.352 |
| $b$ | 0.001 |
| $w$ | 0.063 |
| $q$ | 0.044 |

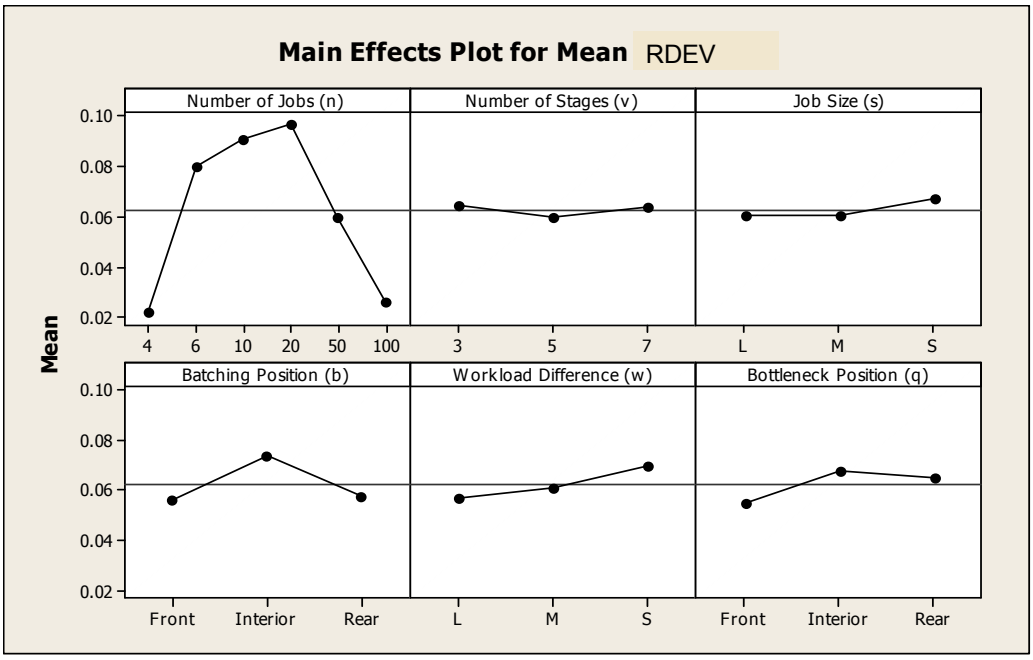



Figure 5-12 Main Effects Plot for Solution Quality of BFD

The main effects plot shows that number of job (*n*) has the greatest impact on the solution quality. For small size instances, solution quality decreases as *n* increases, while for large size instance it is opposite: solution quality increases as *n* increases. Figure 5-13 shows the average RDEV discriminated by number of jobs and Figure 5-14 is the corresponding scatter chart.

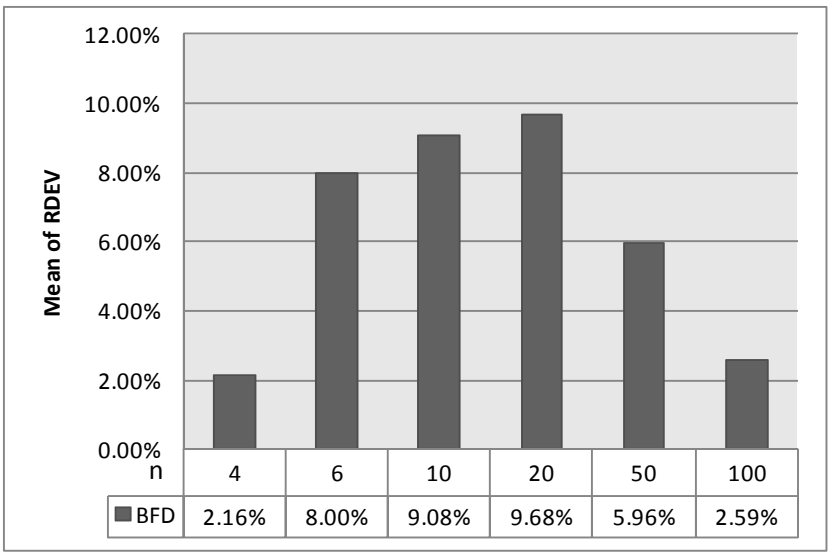



Figure 5-13 Solution Quality of BFD Discriminated by Number of Jobs

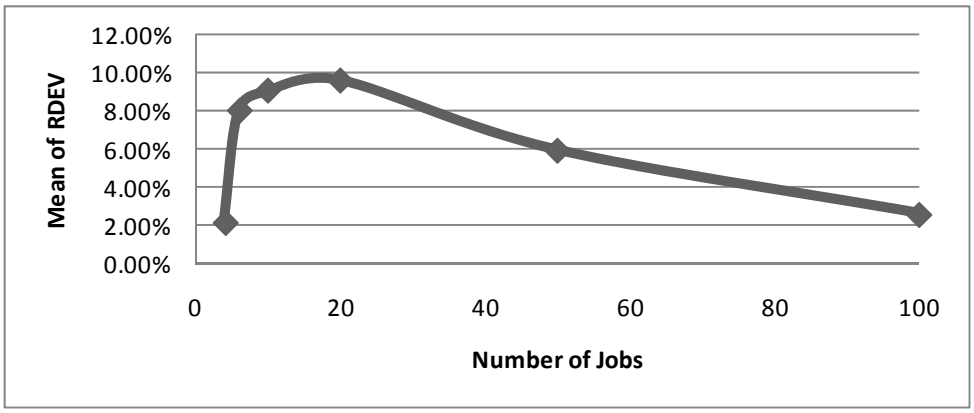



Figure 5-14 Scatter Chart: Solution Quality of BFD by Number of Jobs

The result shows that solution quality decreases dramatically as number of jobs increases from 4 to 6. This is partially due to the benchmark used. As stated in the experiment design section, whenever the optimal solution is available, the optimal solution is used to provide benchmark. Otherwise, a lower bound is used as the benchmark. For 4-job instances, the optimal solution can be found for 88.61% of the

instances. Contrastively, only 27.85% of the 6-job instances are solved to optimality, and none of 10-job instances can be solved. When a lower bound is used to calculate the deviation, it magnifies the deviation. This explains why RDEV drops dramatically as number of jobs increases from 4 to 6.

To exclude the impact of using different benchmark, an extra comparison is performed. In this comparison, lower bounds are used to provide the benchmark in all instances. The deviation from the lower bound is denoted as RDEV*. Figure 5-15 and Figure 5-16 present the RDEV* bar chart and scatter chart respectively. By using lower bound as the benchmark for all instances, the deviation increases from 2.16% to 5.66% for 4-jobs instances, from 8.00% to 8.26% for 6-job instances and remained the same for the rest.

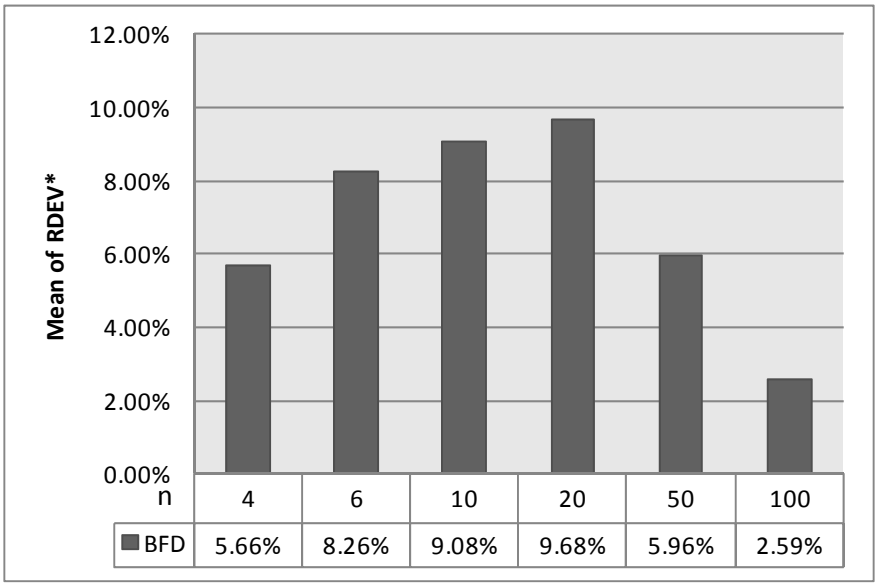



Figure 5-15 RDEV* of BFD Solution Discriminated by Number of Jobs

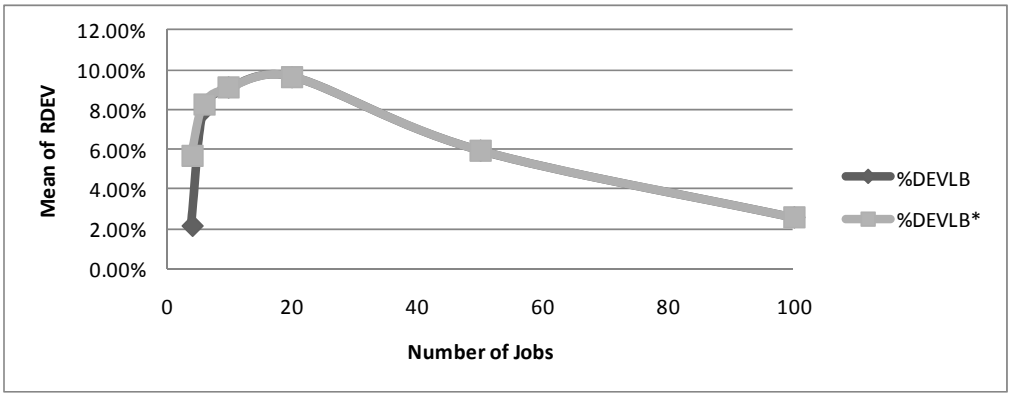



Figure 5-16 Scatter Chart: RDEV* for BFD Solution by Number of Jobs

When number of jobs is greater than or equal to 20, the relative deviation starts decreasing slowly as the number of jobs increases. The decrease is partially due to the change in lower bound quality. As discussed in Chapter 4, the lower bound is getting closer to the optimal $C_{max}$ as number of jobs increases, which leads to the reduction in the deviation.

Based on above analysis, conclusion can be made that number of jobs, *n*, has the greatest impact on solution quality. The solution quality decrease as *n* increases for small size instance ($n \leq 10$), and stabilizes and then slightly increases for median and large size instances ($n \geq 10$). For instances with 50 and 100 jobs, the gaps between BFD solution and the lower bound are as low as 5.96% and 2. 59%, indicating high quality of BFD solution when job number is large.

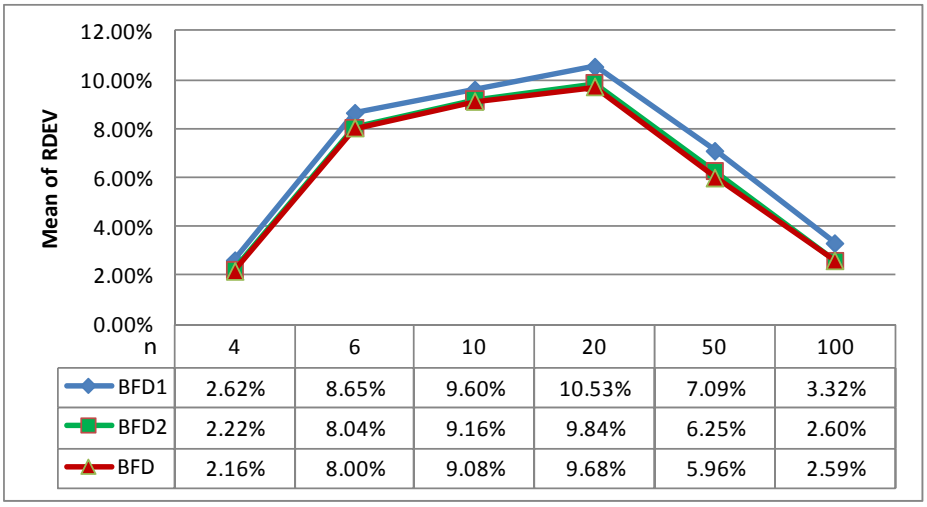
*Impact of Other Factors on the Solution Quality*

The position of batching stage and the position of bottleneck stage also affect the solution quality. BFD produces better results when the batch processing machines or bottleneck located in the first or last stage.

Even though workload difference does not demonstrate statistically significant impact on solution quality, it still has slight impact. Problem instances with higher workload difference tend to get better solutions. This is due to the bottleneck first strategy applied in the heuristic. It gives the bottleneck stage the highest priority to be scheduled, therefore if the workload difference increases, more work will be scheduled with higher priority.

#### 5.8.4.2 BFD vs. Reduced BFDs

The RDEV of solutions produced by BFD and reduced BFDs are presented in Figure 5-17.




| n | 4 | 6 | 10 | 20 | 50 | 100 |
|---|---|---|---|---|---|---|
| BFD1 | 2.62% | 8.65% | 9.60% | 10.53% | 7.09% | 3.32% |
| BFD2 | 2.22% | 8.04% | 9.16% | 9.84% | 6.25% | 2.60% |
| BFD | 2.16% | 8.00% | 9.08% | 9.68% | 5.96% | 2.59% |

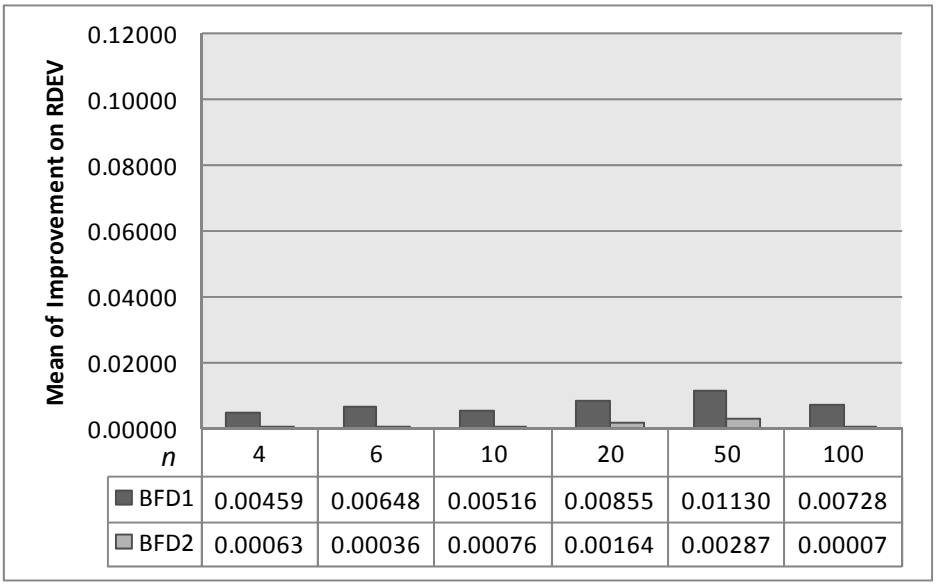
Figure 5-17 Solution Quality of BFD and Reduced BFDs

To explain more clearly the impact of re-scheduling on solution quality, another measurement, IMRDEV, is used. $IMRDEV^{BFD1}$ means the improvement in RDEV on BFD1 after apply full re-scheduling. IMRDEV is calculated as follows:

$$IMRDEV^{BFD*} = RDEV^{BFD*} - RDEV^{BFD} \qquad 5\text{-}14$$

Where BFD* denotes the reduced BFD: BFD1 or BFD2.

Experiment result shows that full re-scheduling reduces the RDEV by 0.0072 in average for BFD1 for all instances, and 0.0011 for BFD2. The breakdown of IMRDEV is given in Figure 5-18.



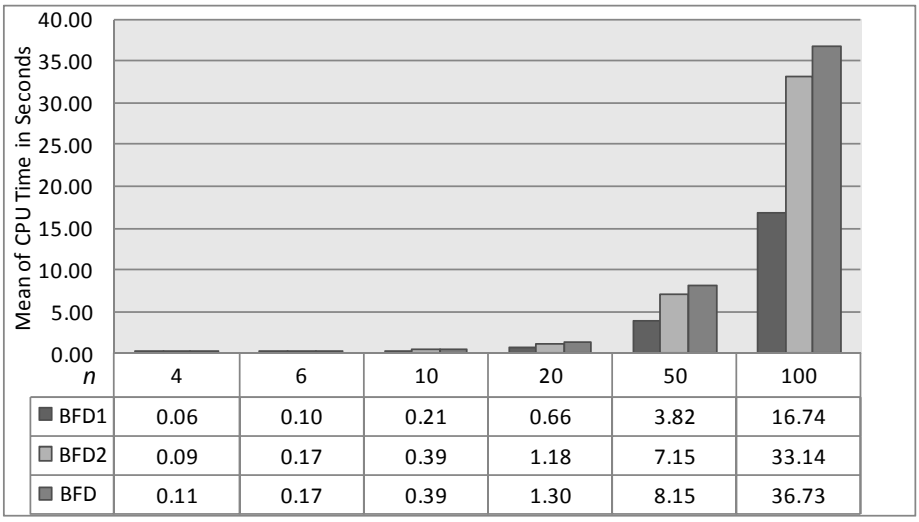

| *n* | 4 | 6 | 10 | 20 | 50 | 100 |
|---|---|---|---|---|---|---|
| BFD1 | 0.00459 | 0.00648 | 0.00516 | 0.00855 | 0.01130 | 0.00728 |
| BFD2 | 0.00063 | 0.00036 | 0.00076 | 0.00164 | 0.00287 | 0.00007 |

Figure 5-18 Average Improvement on Solution Quality by Full Rescheduling

Since re-scheduling is removed completely from BFD1, the algorithm takes much less run time (Figure 5-19), however the solution quality also drops obviously. When re-scheduling is only performed in the last iteration (BFD2), the solution quality is slightly worse than that of BFD, while the run time is also slightly less than that of BFD. The ANOVA results reveal that the difference in solution quality between BFD and BFD1 is statistically significant with a P value equal to 0.035, while the difference between BFD

and BFD2 is not statistically significant, with P=0.738. Even though the improvement on BFD2 is not statistically significant, the algorithm with full re-scheduling consistently outperforms BFD2, with only slight increase in CPU time (Figure 5-19). Therefore, BFD with full re-scheduling is preferable.

The experiment results demonstrate the importance of re-scheduling procedure. Re-scheduling plays an important role in the BFD heuristic by increasing the solution quality, therefore cannot be removed from the algorithm.

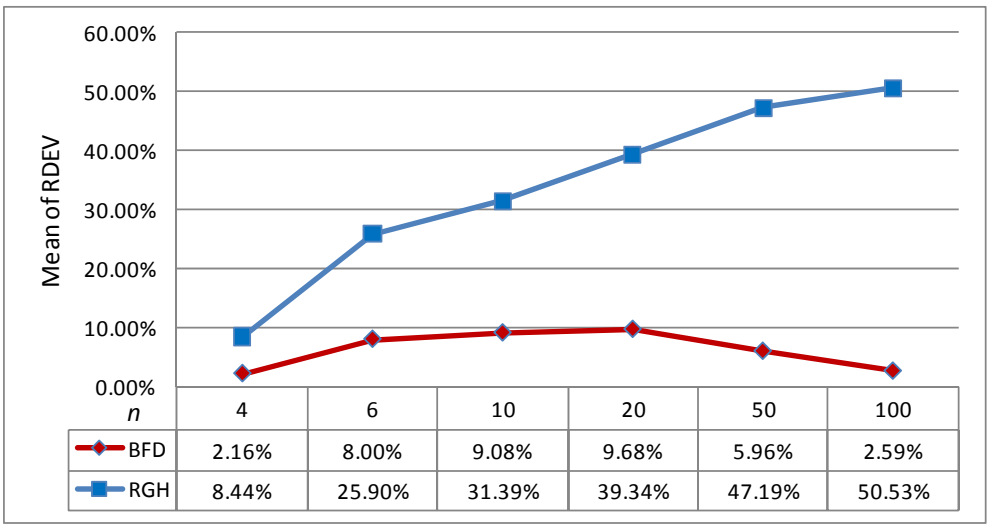



| *n* | 4 | 6 | 10 | 20 | 50 | 100 |
|---|---|---|---|---|---|---|
| BFD1 | 0.06 | 0.10 | 0.21 | 0.66 | 3.82 | 16.74 |
| BFD2 | 0.09 | 0.17 | 0.39 | 1.18 | 7.15 | 33.14 |
| BFD | 0.11 | 0.17 | 0.39 | 1.30 | 8.15 | 36.73 |

Figure 5-19 Average Run Time of in Seconds

5.8.4.3 BFD vs. Randomized Greedy Heuristic

In this section of experiment, each instance is solved by the proposed RGH for 10 times, and the best solution is compared with BFD solution. The summation of 10 run times is considered as the run time for RGH.




| *n* | 4 | 6 | 10 | 20 | 50 | 100 |
|---|---|---|---|---|---|---|
| BFD | 2.16% | 8.00% | 9.08% | 9.68% | 5.96% | 2.59% |
| RGH | 8.44% | 25.90% | 31.39% | 39.34% | 47.19% | 50.53% |

Figure 5-20 RDEV for BFD and RGH Discriminated by Number of Jobs

From Figure 5-20, we can see that the RDEV of RGH increases steadily as *n* increases, and exceeds 50% when *n* reaches 100. BFD shows significant advantage against RGH.

5.8.4.4 BFD vs. List Scheduling with Dispatching Rules

The solution produced by BFD is also compared to the solutions produced by several dispatching rules introduced in the experiment design section. Table 5-18 and Figure 5-21 show the comparison results. The solutions obtained by BFD deviate from the optimal solution or lower bound by 6.25% on average. While the deviations of dispatching rules spread from 14.93% to 20.11%. BFD outperforms all dispatching rules.

Furthermore, BFD solution is compared against the best solution produced by all these dispatching rules, BEST-DSPT. BEST-DSPT has an average deviation of 10.41%, which is higher than that of BFD (6.25%). BFD gives better result than BEST-DSPT.

Table 5-18 Solution Quality of BFD and Dispatching Rules

| n | BFD | BEST_DSPT | LS-ERT | LS-LDT | LS-LPT | LS-LRPT | LS-SPT | Total |
|---|---|---|---|---|---|---|---|---|
| 4 | 2.16% | 6.45% | 12.56% | 9.97% | 14.12% | 10.72% | 11.14% | 9.59% |
| 6 | 8.00% | 12.48% | 22.30% | 16.90% | 23.17% | 18.11% | 20.59% | 17.37% |
| 10 | 9.08% | 13.81% | 24.23% | 19.50% | 25.28% | 19.42% | 23.28% | 19.23% |
| 20 | 9.68% | 15.57% | 27.15% | 20.02% | 28.80% | 20.23% | 23.09% | 20.65% |
| 50 | 5.96% | 9.32% | 20.29% | 13.81% | 21.07% | 12.96% | 15.04% | 14.06% |
| 100 | 2.59% | 4.81% | 14.12% | 10.15% | 14.75% | 8.13% | 8.60% | 9.02% |
| Total | 6.25% | 10.41% | 20.11% | 15.06% | 21.20% | 14.93% | 16.96% | 14.99% |

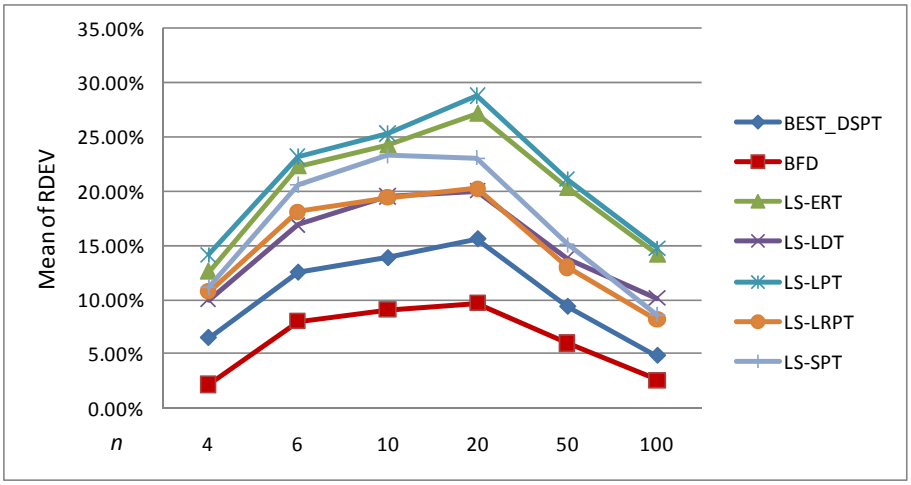
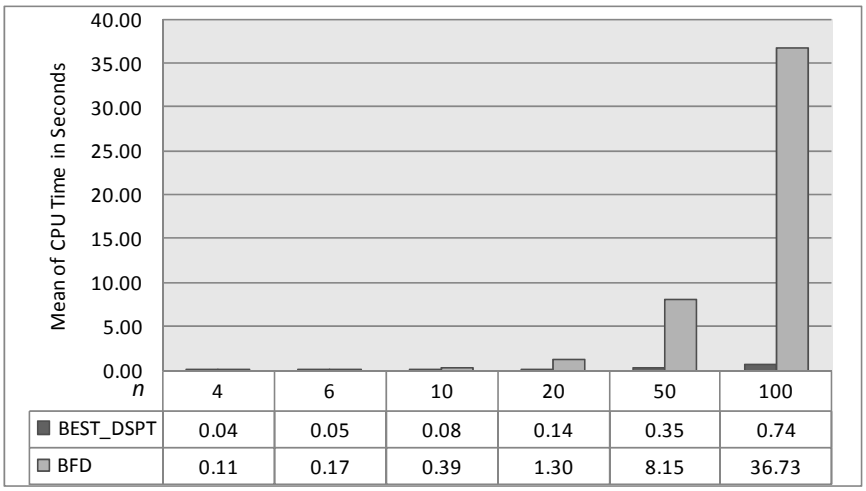



Figure 5-21 Solution Quality of BFD and Dispatching Rules

With regard to the computational cost, dispatching rules are much faster than BFD (Figure 5-22), nevertheless, since BFD requires less than 40 seconds of CPU time in average to solve a large problem instance with 100 jobs, and 8 seconds in average for all problem sizes, the run time required is practical and acceptable. Therefore, BFD is preferable when high solution quality is important. On the other hand, in the situation that time is extremely critical and the quality can be compromised, dispatching rules can serve as fast scheduling algorithms to solve the problem within one second.




| n | 4 | 6 | 10 | 20 | 50 | 100 |
|---|---|---|---|---|---|---|
| BEST_DSPT | 0.04 | 0.05 | 0.08 | 0.14 | 0.35 | 0.74 |
| BFD | 0.11 | 0.17 | 0.39 | 1.30 | 8.15 | 36.73 |

Figure 5-22 Average Run Time of BFD and BEST DSPT in Seconds

## 5.9 Summary

In this chapter, a heuristic approach BFD is proposed for solving the scheduling problem $FF|batch1, s_j|Cmax$. A series of experiments are conducted to evaluate the effectiveness of BFD, which is further evaluated against a set of common algorithms, including a randomized greedy heuristic and five dispatching rules. The results show that the proposed BFD outperforms all these algorithms, and can solve a 100-job instance within a minute with the deviation from the lower bound less than 3% in average. The solution quality increases as number of jobs increases for small size instance ($n \leq 10$), and stabilizes for median and large size problem ($n \geq 10$).

# 6 SOLUTION IMPROVEMENT BY A GENETIC ALGORITHM

A genetic algorithm (GA) with tuned parameters is presented in this chapter to further improve the solution for the problem under study. The solution quality is evaluated against the lower bound and the solution obtained by BFD.

## 6.1 Introduction to Genetic Algorithm

Genetic Algorithm is an adaptive heuristic search algorithm built on the evolutionary ideas of natural selection. By simulating processes in natural system necessary for evolution, GA represents an intelligent exploitation of a random search within a defined search space to solve a problem. GA is often implemented in a computer simulation in which a population of abstract representations (called chromosomes) of candidate solutions to an optimization problem evolves toward better solutions. The evolution usually starts from a population of randomly generated solutions. In each generation, the fitness of every solution in the population is evaluated and a certain number of solutions are stochastically selected based on their fitness. The selected solutions are then further modified to form a new population. The new population is then passed on to the next iteration of the algorithm. The algorithm terminates when either a maximum number of generations has been reached, or a satisfactory fitness level has been met for the population.

A typical GA requires: 1) a genetic representation of the solution domain (or chromosome), 2) genetic operators, and 3) a fitness function evaluating the solution domain. The main property of the genetic representation is that it usually has fixed size which facilitates simple crossover operations. A genetic operator is an operator used in

genetic algorithms to maintain genetic diversity, which is a necessity for the process of evolution. Examples of genetic operators include mutation, crossover, inversion and selection operators. A fitness function is a particular type of objective function that prescribes the optimality of a solution, and is always problem dependent.

A GA has a variety of advantages. It can quickly scan a vast solution set. Bad proposals do not affect the end solution negatively as they are simply discarded. The inductive nature of the GA means that it doesn't have to know any rules of the problem - it works by its own internal rules. This is very useful when the problem is complex defined problems like the one under study in this dissertation.

## 6.2 Proposed Genetic Algorithm

Each iteration of the GA starts with a population of feasible solutions found by heuristics. This initial population is later taken as the initial solution of a local search procedure and the procedure is repeated until some stopping criterion is met. The algorithm flow chart is given in Figure 6-1.

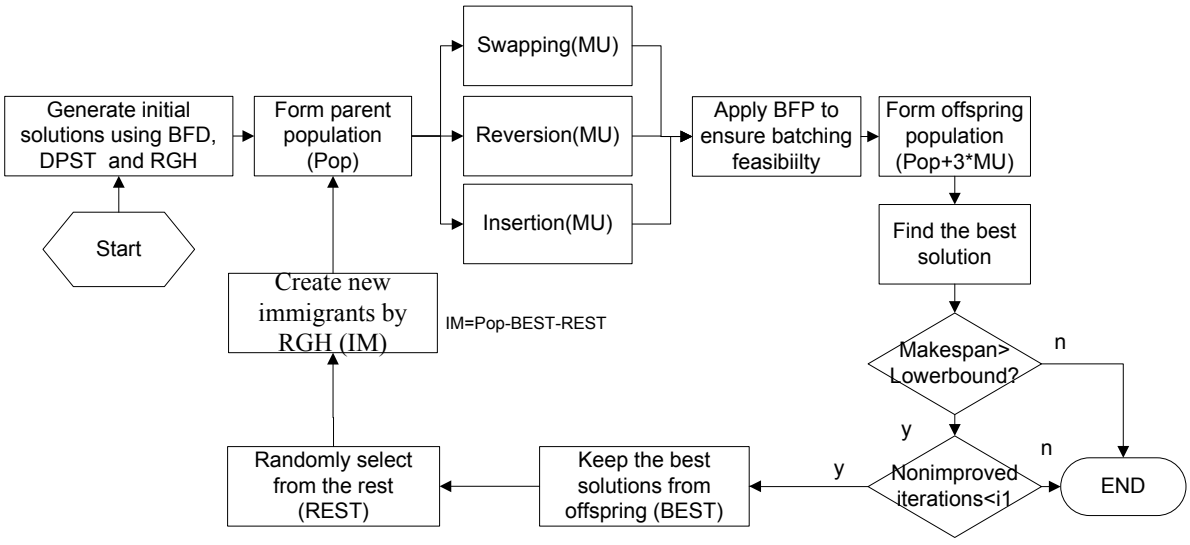


Figure 6-1 Procedure of the Proposed GA

### 6.2.1 Chromosome Representation

The chromosome applied in this algorithm is similar to the 3DGA initially proposed by Amin-Naseri (2009). Each chromosome in the algorithm has a 3D structure. The first dimension indicates the stages, the second dimension shows the number of machines at each stage and the third dimension represents the sequence of each job in that machine. Each variable in the chromosome is called a gene. For instance, the CR(*m, j, i*) represents the gene of the job *j* on machine *m* at stage *i*. It means that to which position on machine *m* the job *j* is assigned at stage *i*. If CR(*m, j, i*) has value zero, it means that job *j* is not assigned to machine *m*. Let's use an example to further illustrate it. Assume a three-stages scheduling problem with five jobs. The first two stages comprise two and one discrete processing machine individually, while the third stage has two batch processing machines. The jobs arrangement is presented in Table 6-1. The structure of the corresponding chromosome is presented in Figure 6-2.

Table 6-1 An Example of Job Sequence on Machines

| Stage | Machine | |
|---|---|---|
| | 1 | 2 |
| 1 | 3→5→1 | 4→2 |
| 2 | 3→1→4→5→2 | -- |
| 3 | 1st Batch: 1,3<br>2nd Batch: 4 | Only one batch: 2,5 |

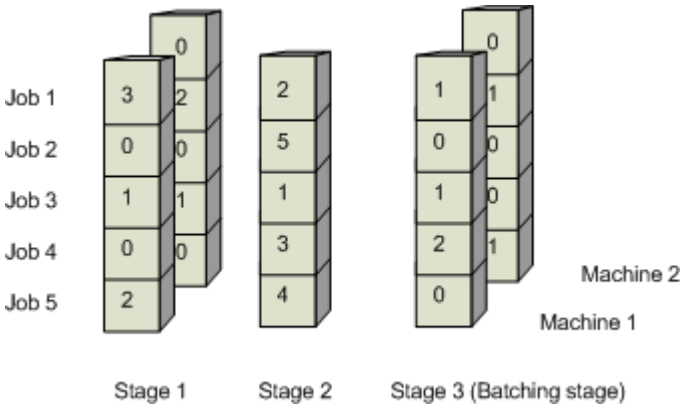



Figure 6-2 3D Chromosome structure

### 6.2.2 Initial Population

Initial population is generated by 1) BFD, 2) five dispatching rules presented in 5.8.1.3, and 3) RGH introduced in 5.8.1.3. Since RGH is a heuristic based on the randomized greedy heuristic, it creates diversity for the initial solutions. The number of initial solutions in the population is a parameter, named *Pop*.

### 6.2.3 Genetic Operators

Three genetic operators are used in this algorithm to perform mutations. For each mutation, a chromosome and a stage in that chromosome is selected randomly. The genetic operators are as follows:

*Swapping*: Two genes are randomly selected at the defined stage. They can be from the same machine or different machines. These two genes are then exchanged with each other.

*Reversion*: Two genes in the same machine are randomly selected. The sequence of genes (only consider the genes with nonzero value) between the two selected genes are reversed.

*Insertion*: A job is randomly selected and inserted to another position or batch, therefore changes the gene of itself as well as those of other jobs related.

The number of iterations of mutations depends on the parameter *MU*, which is a fraction of *Pop*. The maximal amount of offspring chromosomes generated by mutation is then $3MU$.

Crossover operators are also applied to generate offspring. However the preliminary result show that crossover operators do not improve the solution quality, but increase the run time. Therefore, crossover operators are removed from the proposed GA.

### 6.2.4 Batching Feasibility Procedure

Since the mutation operation changes genes randomly, which might causes over size batches, an algorithm named Batching Feasibility Procedure (BFP) is developed to assure the batching feasibility. The BFP borrows the idea from the RBP heuristic presented by Kashan et al. (2008). Some adjustment is applied to fit the problem under study. The BFP algorithm is as follows:

IF there is at least one over size batch exists in the chromosome
THEN
    REPEAT
        Select a batch with capacity violation and the longest batch processing time.
        Select the job with the longest processing time in it.
        IF the selected job fits in any existing batch $b \in B$

THEN

IF among batches in $B$, there are some batches having the batch processing time longer than that of the selected job, and the batch ready time later than that of the selected job. (batches r ⊆ B)

THEN put the selected job in one of the batches in r with the smallest residual capacity.

ELSE put the job in a feasible batch in B with the longest processing time;

END IF

ELSE create a new batch and assign the selected job to it;

END IF

UNTIL a feasible batching plan is obtained

ENDIF

END

### 6.2.5 Selection

After each mutation and the BFP are performed, a new chromosome is generated. If this new chromosome has better fitness than the parent chromosome, then it will be added into the offspring generation. Otherwise it will be abandoned.

Elitism replacement scheme is applied in the GA to select next generation. The good individuals will survive for the next generation and are never lost unless better solutions are found. The elitism replacement is applied as follows: both parent and offspring population are combined into a single population and sorted in a non-increasing order of their makespan. Then, *BEST*Pop* best chromosomes of the combined population are selected as the individuals of the new population for the next generation. In order to avoid premature convergence and to add diversity to the new population, *IM*Pop* new chromosomes generated by RGH are introduced as immigrants to replace the worst chromosomes in the population. Among the remaining individuals, $REST * Pop$

chromosomes are randomly selected to join the next generation of population. *BEST, IM* and *REST* are fractions of the initial population *Pop*. The total number of the chromosomes in the next generation, *BEST+IM+REST,* is equal to *Pop.* Therefore, parameter *IM* is determined by *BEST, REST* and *Pop*, and is not included in the parameter tuning in section 6.3.

### 6.2.6 Stopping criterion

If either of the following two conditions appears, the algorithm is terminated.

1. The best makespan obtained is equal to the lower bound;
2. The makespan is not improved in a number of consecutive generations. This number, called *STOP*, is another parameter.

### 6.2.7 Pseudo code

The pseudo code for the proposed GA approach is shown below:

Let $\boldsymbol{\alpha}$ be the problem instance to be solved and let $\boldsymbol{\tau}$ be the set of solutions in current population. Let $\tau_i$ be a chromosome in $\tau$.
Let Makespan($\tau_i$) be the algorithm that return the makespan of the chromosome $\tau_i$.
Lower bound($\alpha$) returns a lower bound on the makespan for $\alpha$.
BFD($\alpha$) returns a chromosome of solution by applying Bottleneck First Decomposition heuristic.
RGH($\alpha, k$) generates a chromosome by applying randomized greedy algorithm, where k is the size of RCL.
BFP($\tau_i$) returns a feasible chromosome without capacity violation.
Swap($\tau_i$), Insertion($\tau_i$) and Reversion($\tau_i$) return a chromosome in the neighborhood of $\tau_i$ by means of swapping, insertion and reversion respectively. These operators can be applied on genes as well as batches.
Rand(A,n) randomly pick n elements from set A.
$\tau_1 \leftarrow BFD(\alpha)$
For i=*2: Pop*

$\tau_i \leftarrow RGH(\alpha, k)$
End for
$BestCR \leftarrow \emptyset$
$BestMakespan \leftarrow inf$
NonImprove=0
NewCR=0
While *(BestMakespan>LB) & (NonImprove<STOP)*
For *mutation≤MU*
Randomly pick 3 chromosomes $\tau_j$, $\tau_k$ and $\tau_l$ from Ω
NewCR=NewCR+1
$CR\_MU_{NewCR} \leftarrow Swap(\tau_j)$
if $Makespan(CR\_MU_{NewCR}) \geq Makespan(\tau_j)$
NewCR=NewCR-1
End if
NewCR=NewCR+1
$CR\_MU_{NewCR} \leftarrow Insertion(\tau_k)$
if $Makespan(CR\_MU_{NewCR}) \geq Makespan(\tau_k)$
NewCR=NewCR-1
End if
NewCR=NewCR+1
$CR\_MU_{NewCR} \leftarrow Reversion(\tau_l)$
if $Makespan(CR\_MU_{NewCR}) \geq Makespan(\tau_l)$
NewCR=NewCR-1
End if
End for
$OffSpring \leftarrow \omega \cup CR\_MU$
$OffSpring\_S \leftarrow sort\ OffSpring\ in\ nondecreasing\ makespan$
$\mu \leftarrow Best\ chromosome\ in\ OffSpring$
If $Makespan(\mu) < BestMakespan$
$BestMakespan \leftarrow Makespan(\mu)$
$BestCR \leftarrow \mu$
$NonImprove \leftarrow 0$
Else
$NonImprove \leftarrow NonImprove + 1$
End if
$\tau(1{:}\,BEST) \leftarrow OffSpring\_S(1{:}\,BEST)$
$OffSpring\_S(1{:}\,BEST) \leftarrow \emptyset$
$\tau(BEST + 1{:}\,BEST + REST) = Rand(OffSpring\_S, REST)$
For m=1: *Pop-BEST-REST*

$\tau(BEST + REST + M) \leftarrow RGH(\alpha, k)$

End for

End while

There are a number of parameters which affects the algorithm performance. These parameters include:

1. *Pop*, the population size, a fraction of n. It increases as the problem size increases;
2. *STOP*, the maximal number of consecutive iterations allowed without improvement;
3. *K*, the maximal number of elements allowed in RCL;
4. *MU*, mutation rate;
5. *BEST*, the percentage of the top candidates passed on to the next generation;
6. *REST*, the percentage of population chosen randomly from the remaining candidates.

## 6.3 Parameters Tuning

A full factorial experiment is conducted to determine the appropriate values of the parameters. The factors and levels tested are shown in the Table 6-2. All the combinations of the parameter values are tested on 20 instances of 10-job and 20 instances of 20-job, since the solution obtained by BFD for 10-job and 20-job instances has the largest deviation from the lower bound. The deviation between the GA solution and the lower bound is defined as the dependent variable.

$$\boldsymbol{RDEV} = \frac{\boldsymbol{BestMakespan - Lowerbound}}{\boldsymbol{Lowerbound}} \times \mathbf{100\%} \qquad 6\text{-}1$$

Table 6-2 Parameters in GA

| Parameter | Description | Level |
|---|---|---|
| *Pop* | Population size | [*n* 2*n* 3*n*] |
| *STOP* | Maximal number of consecutive iterations allowed without improvement | [12 50 100] |
| *K* | The maximal number of elements allowed in RCL when applying RGH | [1 3 5] |
| *MU* | Mutation rate | [0.3 0.5 0.7] |
| *BEST* | The percentage of the population chosen from the top to next generation | [0.1 0.3 0.5] |
| *REST* | The percentage of population chosen from the remaining offspring population | [0.1 0.3 0.5] |

The analysis of variance conducted with a significance level of 5% is presented in Table 6-3. The analysis revealed that all parameters except for *REST* have a significant effect on the solution quality. Figure 6-3 shows the main effects plot for each parameter. The selected parameter values are as follows: *Pop*=*3n*, *K*=3, *MU*=0.7, *BEST*=0.3 and *REST*=0.3. As for the parameter *STOP*, when *STOP*=100, GA produced the best solution. However, for 50-job instances and 100-job instances, the algorithm requires tremendous run time (over 2 hours for each instance). Therefore, for 4-job, 6-job, 10-job and 20-job instances, *STOP* is set as 100, while for 50-job and 100-job instances, *STOP* is set as 12 and 50, respectively, to reduce run time. Since the solution produced by BFD is close to optimal in average when number of jobs are greater than 50, there is not as much

improving potential by GA. Therefore the value of *STOP* will not cause a significant impact on the solution quality for 50-job and 100-job instances. The value of the non-significant parameter, *REST*, is set so that the run time is minimized.

Table 6-3 ANOVA Table for GA Parameter Selection

| Parameter | P Value |
|---|---|
| *Pop* | 0.0000 |
| *STOP* | 0.0000 |
| *K* | 0.0418 |
| *MU* | 0.0001 |
| *BEST* | 0.0457 |
| *REST* | 0.5765 |

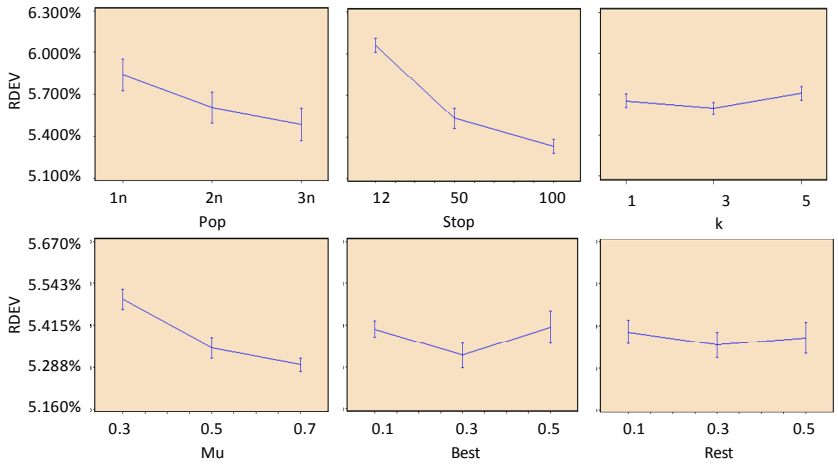



Figure 6-3 Main effects plot for GA parameter selection

## 6.4 Computational Result

A full factorial design of experiments based on the same set of 1458 random generated instances from chapter 5 is conducted in order to determine the impact of each factor on the quality of the proposed GA approach. The experiment used the same factors as defined in Table 5-16, including number of jobs (*n*), number of stages (*v*), size of jobs (*s*), position of batching stage (*b*), position of bottleneck stage (*q*) and workload difference (*w*). Each problem instance is solved by GA 3 times and the average result is reported.

The percentage of deviation from lower bound or optimal solution calculated by equation 6-2 is applied to evaluate the performance.

$$\boldsymbol{RDEV} = \frac{(C_{max}^{GA} - \boldsymbol{LB})}{C_{max}^{GA}} \times \mathbf{100}\% \qquad 6\text{-}2$$

Table 6-4 ANOVA Table for RDEV of GA

| Factors | P Value |
|---|---|
| *n* | 0.000 |
| *v* | 0.698 |
| *s* | 0.377 |
| *b* | 0.000 |
| *w* | 0.047 |
| *q* | 0.017 |

Table 6-4 and Figure 6-4 present the result of analysis of variance. Four out of six factors, problem size, batching position, bottleneck position and workload difference, are

significant at a 5% confidence level. On the other hand, job size and number of stages do not have significant impact.

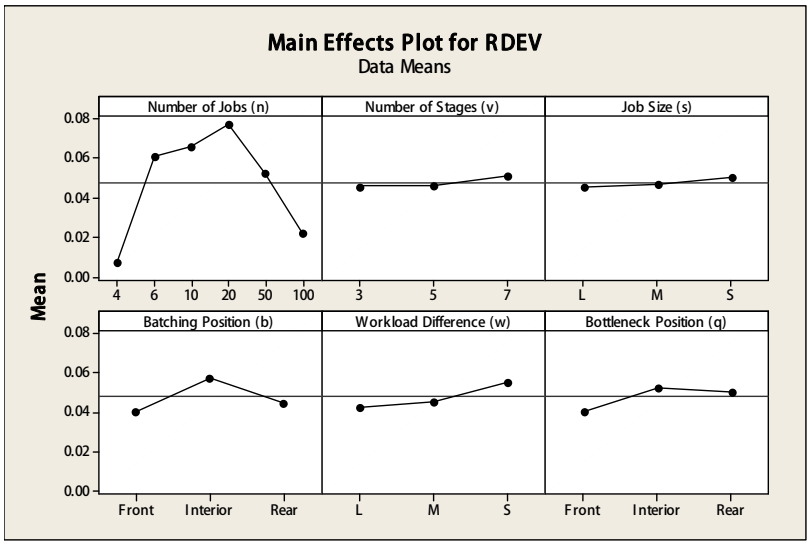



Figure 6-4 Main Effects Plot for RDEV of GA

The algorithm performs better when number of jobs is small or large. The instance with batching machines at the first or the last stage is easier to solve than at the interior stages. Besides, the solution quality is also better when the bottleneck locates at the first or last stage. The problem instances with higher workload difference result in better solutions. The comparison of *RDEV* between BFD and GA is given in Figure 6-5.

To further analyze the improvement on solution by GA, two more measurements, *IMDEV* and *RIMDEV* are used. While $IMDEV$ means the improvement in *RDEV* on BFD by GA, $RIMDEV$ represents the percentage of improvement. These measurements are calculated as follows:

$$\boldsymbol{IMDEV = RDEV^{BFD} - RDEV^{GA}} \qquad 6\text{-}3$$

$$\%IMDEV = \frac{RDEV^{BFD} - RDEV^{GA}}{RDEV^{GA}} \times 100\% \qquad 6\text{-}4$$

Computational results show that GA reduces the *RDEV* by 1.47% unit in average, and the relative improvement on RDEV is 19.95%. Figure 6-6 describes the improvement on *RDEV* discriminated by problem size, and Figure 6-7 presents the relative improvement of *RDEV* discriminated by problem size.

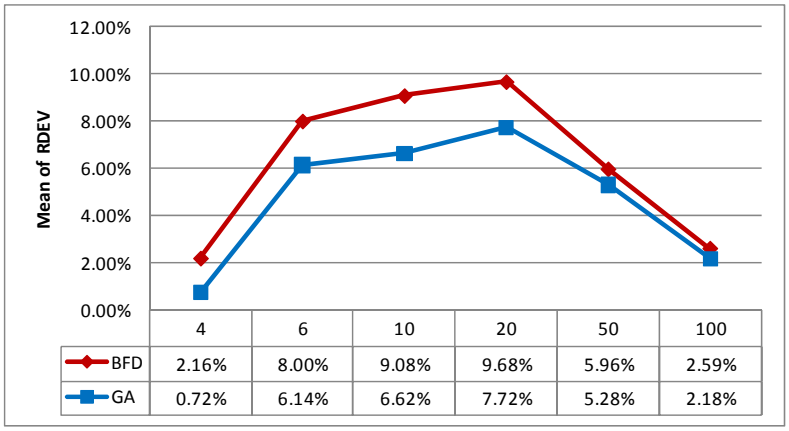



Figure 6-5 Average RDEV of BFD and GA

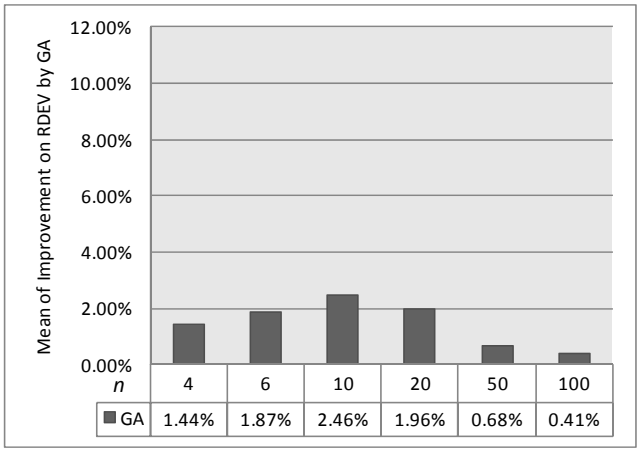



Figure 6-6 Average Improvement in RDEV by GA

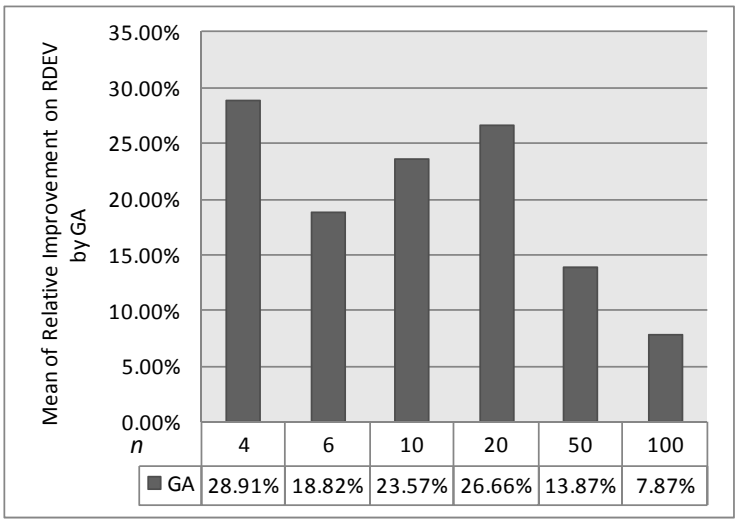
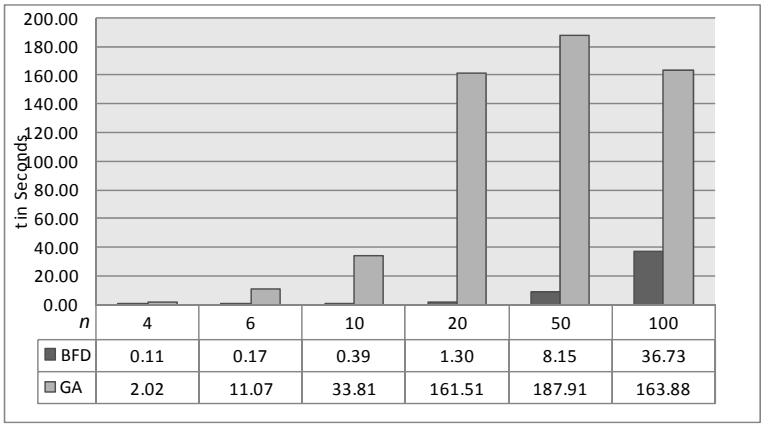



Figure 6-7 Percentage of Improvement on RDEV by GA

The GA algorithm is coded in Matlab 7.04 and the experiments are run on a desktop computer with a 2.21 GHz *Intel Core 2 Duo* ® processor with 2 GB of RAM. Figure 6-8 shows the average run times of the GA algorithm compared to the run times of BFD discriminate by problem size. The results show that the run time of GA increases dramatically as problem size increases, and is significantly longer than that of BFD. Run times for 50-job and 100-job instances are not significant higher than that of 20-job instances. This is because the value of parameter, *STOP,* are reduced from 100 to 50 for 50-job instances, and to 12 for 100-job instances, as explained in section 6.3.

In summary, the proposed GA algorithm improves the solution of BFD by around 20% in a negligible time when problem size is less than 50 jobs. When the problem size reaches 50 jobs, the computational time required is increased dramatically, and the improvement over BFD is very limited since the average quality of BFD solutions for 50-

job and 100-job instances are higher than those of smaller-size instances and leave limited space for improvement. As a result, GA can be used to improve solution for small and median size instances ($n < 50$), while for solving large instances BFD is more efficient and preferable.




| n | 4 | 6 | 10 | 20 | 50 | 100 |
|---|---|---|---|---|---|---|
| BFD | 0.11 | 0.17 | 0.39 | 1.30 | 8.15 | 36.73 |
| GA | 2.02 | 11.07 | 33.81 | 161.51 | 187.91 | 163.88 |

Figure 6-8 Average Run Time of GA vs. BFD in Seconds

# 7 CONCLUSION AND FUTURE RESEARCH

This chapter re-examines the results in light of the original questions proposed in Chapter 1, presents the summary of findings and conclusions, and concludes with an exploration of possible future directions for the research.

## 7.1 Overview of the Problem under Study

In this dissertation, research is conducted on a scheduling problem, $FF|batch1, s_j|Cmax$, which is to minimize the makespan on the hybrid flow shop with parallel batch-processing machines in one of the stages to process jobs with arbitrary sizes. A mathematical model is developed to formulate the problem, heuristics are proposed to solve the problem and extensive computational experiments are performed to evaluate the performance of proposed heuristics.

## 7.2 Findings

A mixed-integer-linear-programming model is developed to formulate the problem under study, $FF|batch1, s_j|Cmax$. Randomly generated problem instances are solved by the commercial solver AMPL/CPLEX. The result reveals that problem size, number of stage and the position of batching stage have significant impact on computational time. The run time required to solve the problem to optimality increases dramatically when job number or stage number increases. Within two hours, the solver can only solve a problem instance with up to 6 jobs. As the problem size increases to beyond 6 jobs, the computational cost of solving the problem to optimality is too large to be used in practical implementations. It is also discovered that a problem instance with

batching machines in an interior stage requires more time to find the optimal solution than the one with batching machines in the first or last stage.

Since optimal solution cannot be found within reasonable time on any problem instance with more than 6 jobs, a lower bound is required to serve as the benchmark for heuristic performance evaluation. For this purpose, a tight lower bound is proposed. This lower bound is derived by a three-level lower bound procedure. 1010 instances with 4 jobs, 5 jobs or 6 jobs are tested to evaluate the lower bound quality. The lower bound obtained is compared with the optimal solution solved by AMPL/CPLEX solver. The result shows that the proposed lower bound procedure generates tight lower bounds, and the tightness increases substantially as the number of jobs increases. On the other hand, the number of stages shows no great impact on lower bound quality. Besides, the lower bound of small problem instances with less than 20 jobs can be further improved by a lifting procedure by 0.51% in average.

To be able to solve moderate and large size problem instances efficiently, a heuristic approach called bottleneck-first-decomposition heuristic (BFD) is proposed. The BFD heuristic decomposes $FF|batch1, s_j|Cmax$ into three sub-problems: upstream hybrid flow shop, parallel batch processing stage, and downstream hybrid flow shop, and schedules the sub-problems one by one by identifying and scheduling the bottleneck one at a time. Two heuristics are proposed to solve sub-problems: one is LSD for solving $P_m|r_j, q_j, s_j, batch|C_{\max}$' and the other one is BA-Jackson for solving $FF_m|r_j, q_j|C_{\max}$'. At the batch-processing stage, batches are formed by a list scheduling with the dispatching rule of largest processing time + delivery time $(p_j + q_j)$. At the discrete-processing

stages, jobs are scheduled using a list scheduling with the largest delivery time ($q_j$), breaking tie with LPT rule. Since jobs have un-equal arrival times at each stage, a delay strategy is applied to identify a pool of candidate jobs for next machine or for forming a batch. For batch-processing, the delay strategy allows a job with late arrival time to join the batch and hence improve the utilization of current batch space. For discrete-processing, it allows a job with late arrival time but large delivery time to have higher priority to be processed, therefore reduces the chance of being delayed by this job at remaining stages. The interaction between stages is modeled as job arrival times ($r_j$) and delivery times ($q_j$). $r_j$ and $q_j$ for a stage are determined by the job precedence relationship at previous stages and remaining stages, respectively, and form the input to schedule the current stage. Solutions obtained by BFD are deviated from the optimal solution or lower bound by 6.87% in average for all problem instances tested, and BFD consistently outperform the randomized greedy heuristic, various dispatching rules as well as the best solution produced by the dispatching rules. The number of jobs, *n*, has the greatest impact on solution quality. The solution quality decrease as *n* increases for small size instance ($n \leq 10$), and stabilizes and then slightly increases for median and large size instances ($n \geq 10$). BFD produces better results when the batch processing machines or bottleneck located in the first or last stage. Problem instances with higher workload difference tend to get better solutions. This is due to the bottleneck first strategy applied in the heuristic. It gives the bottleneck stage the highest priority to be scheduled, therefore if the workload difference increases, more work will be scheduled with higher priority. Results also show that re-scheduling procedure has a significant effect on both solution quality and computation time by decreasing the deviation by

4.65% and doubles the run time. Since increment in run time is limited and acceptable, the re-scheduling procedure is preferable in order to increase the solution quality. Overall, the results indicate that decomposition methods such as that described in this dissertation offers significant potential as solution procedures for complicate practical scheduling. The sub-problem modeling and sub-problem solution approaches proposed in this research perform well as components for the decomposition-based heuristic.

To further improve the heuristic solution, a genetic algorithm GA is developed. In The proposed GA algorithm effectively improves the solution of BFD in a negligible time when problem size is less than 50 jobs. When the problem size reaches 50 jobs, the computational time required is increased dramatically with limited improvement. As a result, GA can be used to improve solution for small and median size instances ($n < 50$), while for solving large instances BFD is preferable.

Figure 7-1 and Figure 7-2 and present a comparison of the performance for all the heuristic approaches proposed in this research. The two measurements are the deviation from the lower bound, *RDEV*, and run times.

Based on above performance evaluation, a selecting criterion of solution approaches is presented as follows: For problems with median and small number of jobs (n<50), the meta-heuristic approach GA is the most effective approach to solve the problem, while for problems with large number of jobs (n>=50), BFD can yield good results within reasonable time.

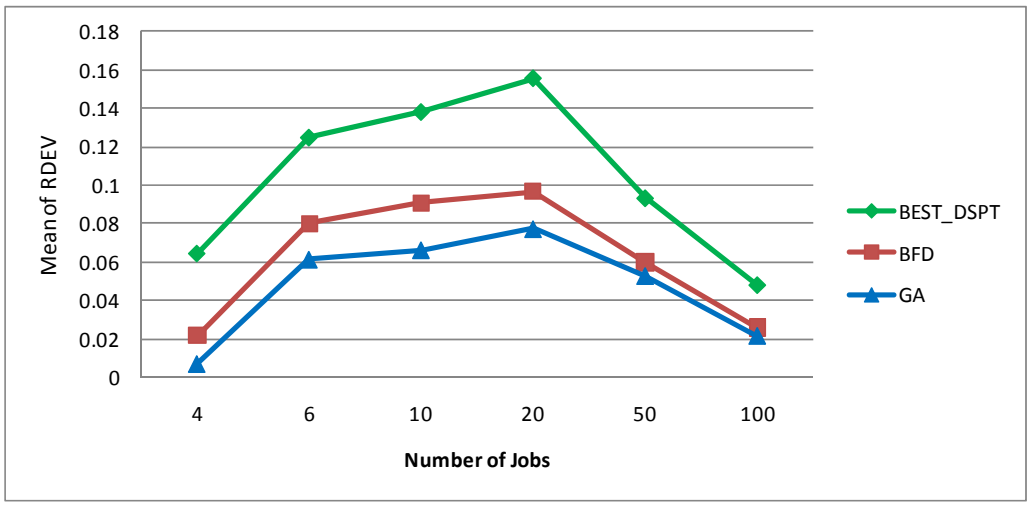



Figure 7-1 Solution Quality of Different Algorithms

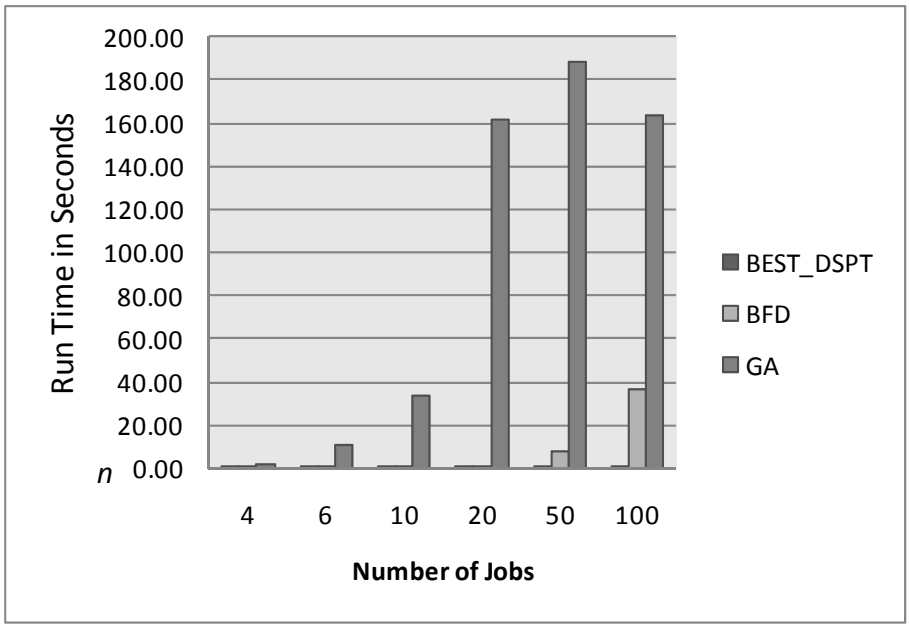



Figure 7-2 Average Run Times in Seconds for All Algorithms

## 7.3 Future Research

The development of an effective decomposition procedure also requires effective solution procedures for the sub-problems. To reduce computational cost, this research applies heuristics to get approximate solutions for the sub-problems, which reduce the run time in a great sense, while caused unavoidable impact on the final solution quality. Future research may focus on developing exact solutions for the sub-problems, for example branch and bound, to improve the solution quality. However, the computational time can significantly increase.

For the same purpose of reducing computational cost, instead of decomposing the shop into single stage scheduling problem, the proposed heuristic decomposes the problem into three sub-problems each containing one or more stages, so that re-scheduling time is reduced. Future research may stick to the original decomposition method used by the shifting bottleneck heuristic and consider each stage as a sub-problem.

Another important issue which needs to be addressed for the BFD heuristic to be useful in practice is that of how to handle uncertainties occurring on the shop floor. The design of effective heuristic for the case of stochastic arrivals, in which the arrival of the job is not known in advance, is also an opportunity for further research.

To simplify the problem, the capacity of the batching machines and physical size of jobs are assumed one-dimension in this research. The future research can change this assumption to 2-dimension or 3-dimemsion.

Objectives other than Cmax, i.e. due date related objectives and multiple objectives can also be investigated. This dissertation focuses on minimizing the

makespan since it can help to increase the utilization of machine and labor resource, reduce the lead time of a manufacturing order and therefore improve the manufacturer's strength of competition. Other objectives besides Cmax can also be important, for example minimizing maximum lateness or minimizing the number of late jobs. These due-date related objectives are especially important when jobs to process have a tight due date or the capacity of the resource cannot meet the demand.

LIST OF REFERENCES

Acero-Domínguez, M. J. and C. D. Paternina-Arboleda (2004). SCHEDULING JOBS ON A K-STAGE FLEXIBLE FLOW SHOP USING A TOC-BASED (BOTTLENECK) PROCEDURE. the 2004 Systems and Information Engineering Design Symposium.

Adams, J., E. Balas, et al. (1988). "The Shifting Bottleneck Procedure for Job Shop Scheduling." Management Science 34(3): 391-401.

Adler, L., N. Fraiman, et al. (1993). "BPSS: a scheduling support system for the packaging industry." Operations Research 41: 641-648.

AHMADI, J. H., R. H. AHMADI, et al. (1992). "BATCHING AND SCHEDULING JOBS ON BATCH AND DISCRETE PROCESSORS." Operations Research 40(4): 750 - 763

Amin-Naseri, M. R. and M. A. Beheshti-Nia (2009). "Hybrid flow shop scheduling with parallel batching." International Journal of Production Economics 117: 185-196.

Arthanari, T. S. and K. G. Ramamurthy (1971). "An extension of two machines sequencing problem." Opsearch 8: 10-22.

Baker, K. R. (1974). Introduction to Sequencing and Scheduling, Wiley.

Balas, E., J. K. Lenstra, et al. (1995). "One Machine Scheduling with Delayed Precedence Constraints." Management Science 41(94-109).

Bellanger, A. and A.Oulamara (2009). "Scheduling hybrid flow shop with parallel batching machines and compatibilities." Computers &OperationsResearch 36(1982--1992).

Bellanger, A. and A. Oulamara (2008). Exact Method For Hybrid Flowshop With Batching Machines And Tasks Compatibilities. The Eleventh International Workshop on Project Management and Scheduling - PMS 2008.

Black, P. E. (2005). "greedy algorithm" in Dictionary of Algorithms and Data Structures, U.S. National Institute of Standards and Technology.

Brah, S. A. and J. L. Hunsucker (1991). "Branch and bound algorithm for the flowshop with multiple processors." European Journal of Operational Research 51: 88-99.

Brah, S. A. and L. L. Loo (1999). "Heuristics for scheduling in a flow shop with multiple processors." European Journal of Operational Research 113(1): 113-122.

Campbell, H. G., R. A. Dudek, et al. (1970). "A heuristic algorithm of the n-job, m-machine sequencing problem." Manag. Sci. 16: b630-b637.

Caricato, P., A. Grieco, et al. (2007). "Tsp-based scheduling in a batch-wise hybrid flow-shop." Robotics and Computer-Integrated Manufacturing 23: 234-241.

Carlier, J. (1987). "Scheduling jobs with release dates and tails on identical machines to minimize the makespan." European Journal of Operational Research 29: 298-306.

Carlier, J. and E. Pinson (1998). "Jackson's pseudo preemptive schedule for the Pm/rj,qj/Cmax scheduling problem." Annals of Operations Research 83(1): 41-58.

Chang P-Y., D. P., et al. (2004). "Minimizing makespan on parallel batch processing machines." International Journal of Production Research 42: 11-20.

Chaudhry, S. S. and W. Luo (2005). "Application of genetic algorithms in production and operations management: a review." International Journal of Production Research 43(19): 4083.

Chen, C.-L. and Chuen-LungChen (2008). "Bottleneck-based heuristics to minimize tardy jobs in a flexible flow line with unrelated parallel machines." International Journal of Production Research 46(22): 6415-6430.

Chen, C.-L. and Chuen-LungChen (2009). "A bottleneck-based heuristic for minimizing makespan in a flexible flow line with unrelated parallel machines." Computers &OperationsResearch.

Chen, Y. C. and C. E. Lee (1998). "Bottleneck-based group scheduling in a flow line cell." International Journal of Industrial Engineering -Application and Practice 5: 288-300.

Cheng, J., Y. Karuno, et al. (2001). "A shifting bottleneck approach for a parallel-machine flowshop scheduling problem." Journal of Operations Research 44(2).

Chiang, T.-C., H.-C. Cheng, et al. (2008). An efficient heuristic for minimizing maximum lateness on parallel batch machines. Eighth International conference on Intelligent Systems Design and Applications.

Chung, S. H., Tai, Y.T and Pearn, W.L., (2008). "Minimising makespan on parallel batch processing machines with non-identical ready time and arbitrary job sizes." International Journal of Production Research.

Conway, R. (1997). "Comments on an exposition of multiple constraint scheduliing." Production and Operation Management 6: 23-24.

D., G. (1984). "Bounds for naive multiple machine scheduling with release times and deadlines." Journal of Algorithms 5(1-6).

Damodaran, P. and P.-Y. Chang (2007). "Heuristics to minimize makespan of parallel batch processing machines " The International Journal of Advanced Manufacturing Technology 37: 1005-1013.

Damodaran, P., N. S. Hirani, et al. (2008). "Scheduling identical parallel batch processing machines to minimise makespan using genetic algorithms." European Journal of Industrial Engineering 3: 187 - 206.

Damodaran, P., M.C. Velez-Gallego (2008). "Minimising makespan on parallel batch-processing machines with unequal job ready times." Under review.

Ding, F. Y. and D. Kittichartphayak (1994). "Heuristics for scheduling flexible flow lines." Computers and Industrial Engineering 26: 27-34.

Draidi, F. (2004). Parameterless genetic algorithms: Review, comparison and improvement, Concordia University. M.A.Sc: 126.

El-Mihoub, T. A., A. A. Hopgood, et al. (2006) "Hybrid Genetic Algorithms: A Review." Engineering Letters Volume, DOI:

Gharbi, A. and M. Haouari (2007). "An approximate decomposition algorithm for scheduling on parallel machines with heads and tails." Computers & Operations Research 34: 868-883.

Goldratt, E. and R. Fox (1986). The Race. New York, North River Press.

Goldratt, T. and J. Cox (1992). the goal, North River Press.

Gourgand, M., Grangeon, N., Norre, S., (1999). "Metaheuristics for the deterministic hybrid flowshop problem." Proceedings of International Conference on Industrial Engineering and Production management: 136-145.

Guinet, A. G. P. and M. M. Solomon (1996). "Scheduling hybrid flowshops to minimize maximum tardiness or maximum completion time." International Journal of Production Research 34(6): 1643-1654.

Gupta, J. N. D. (1972). "Heuristic algorithms for multistage flowshop scheduling problem." AIIE Transactions 4: 11-18.

Gupta, J. N. D., A. M. A. Hariri, et al. (1997). "Scheduling a two-stage hybrid flow shop with parallel mahines at the first stage." Annals of Operation Research 69: 171-191.

Gupta, J. N. D., K. Kruger, et al. (2002). "Heuristics for hybrid flow shops with controllable processing times and assignable due dates." Computers and Operations Research 29(10): 1417-1439.

Gupta, J. N. D. and E. A. Tunc (1991). "Schedules for a two-stage hybrid flowshop with parallel machines at the second stage." International Journal of Production Research 29(7): 1489-1502.

Gupta, J. N. D. and E. A. Tunc (1994). "Scheduling a two-stage hybrid flowshop with separable setup and removal times." European Journal of Operational Research 77: 415-428.

Haouari, M. and A. Charbi (2004). "Lower bounds for scheduling on identical parallel machines with heads and tails." Annals of Operations Research 129: 187-204.

Haouari, M., L. Hidri, et al. (2006). "Optimal scheduling of a two-stage hybrid flow shop.". Meth. Oper. Res. 64: 107-124.

Hart, J. P. and A. W. Shogan (1987). "Semi-greedy heuristics: An empirical study." Operations Research Letters 6: 107-114.

Hoogeveen, J. A., Lenstra, J.K., Veltman, B., (1996). "Preemption scheduling in a two-stage multiprocessor flowshop is NPhard." European Journal of Operational Research 89: 172-175.

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

J, C. and N. E (2000). "An exact method for solving themultiprocessor flowshop." RAIRO-Oper Res 34(1-25).

Jin, Z. H., Ohno, K., Ito, T., Elmaghraby, S.E., (2002). "Scheduling hybrid flowshops in printed circuit board assembly line." Production and Operation Management 11(1): 216-230.

Kashan, A. H., B. Karimi, et al. (2008). "A hybrid genetic heuristic for scheduling parallel batch processing machines with arbitrary job sizes." Computers & Operations Research 35: 1084-1098.

Kim, Y.-D., B.-J. Joo, et al. (2009). "Heuristics for a two-stage hybrid flowshop scheduling problem with ready times and a product-mix ratio constraint." Journal of Heuristics 15: 19-42.

Kurz, M. E. and R. G. Askin (2003). "Comparing scheduling rules for flexible flow lines." International Journal of Production Economics 85: 371-388.

Lee, C.-Y., R. Uzsoy, et al. (1992). "Efficient Algorithms for Scheduling Semiconductor Burn-In Operations." INFORMS 40: 764-775.

Lee, C. Y. and R. Uzsoy (1999). "Minimising makespan on a single batch processing machine with dynamic job arrivals." International Journal of Production Research 37(1): 219–236.

Lee, G.-C., Y.-D. Kim, et al. (2004). "Bottleneck-focused scheduling for a hybrid flowshop." International Journal of Production Research 42(1): 165-181.

Li, S., G. Li, et al. (2004). Minimizing Maximum Lateness on Identical Parallel Batch Processing Machines.

Lin, B. M. T. and A. A. K. Jeng (2004). "Parallel-machine batch scheduling to minimize the maximum lateness and the number of tardy jobs." International Journal of Production Economics 91(2): 121-134.

M, P. (1995). Branch and bound method for the multiprocessor jobshop and flowshop scheduling problem. Department of Computer Science, University of Copenhagen.

Martello, S. and P. Toth (1990). Knapsack problems: Algorithms and computer implementations, John Wiley & Sons.

Narasimhan, S. L. and P. M. Mangiameli (1987). "A comparison of sequencing rules for a two-stage hybrid flow shop." Decision Sciences 18: 250-265.

Narasimhan, S. L. and S. S. Panwalker (1984). "Scheduling in a two-stage manufacturing process." International Journal of Production Research 22(4): 555-564.

Neron, E., P. Baptiste, et al. (2001). "Solving hybrid flow shop problem using energetic reasoning and global operations." Omega 29(6): 501-511.

Nowicki, E., Smutnicki, C., (1998). "The flowshop with parallel machines: A tabu search approach." European Journal of Operational Research 106(226): 253.

O, M. and P. Y (2000). "A branch and bound algorithm for the hybrid flowshop." International Journal of Production Economics 64(113-125).

Orhan Engin, A. D. (2004). "A New Approach to Solve Hybrid Flow Shop Scheduling Problems by Aritficial Immune System." Future Generation Computer Systems 20: 1083-1095.

Oulamara, A., G. Finke, et al. (2009). "Flowshop scheduling problem with a batching machine and task compatibilities." Computers and Operations Research 36(2): 391-401.

Ovacik, I. M. and R. Uzsoy (1997). Decomposition Methods for Complex Factory Scheduling Problems. Boston, Kluwer Academic Publishers.

Paternina-Arboleda, C. D., J. R.Montoya-Torres, et al. (2008). "Scheduling jobs on a k-stage flexible flow-shop." Annals of Operation Research 164: 29-40.

Portmann, M., A. Vignier, et al. (1998). "Branch and bound crossed with GA to solve hybrid flowshops." European Journal of Operational Research 107: 389-400.

Portmann, M. C., A. Vignier, et al. (1998). "Branch and bound corssed with GA to solve hybrid flowshops." European Journal of Operational Research 107(2): 389-400.

Rajendran, C. and D. Chaudhuri (1992). "Scheduling in n-job m stage flowshop with parallel processors to minimize makespan." International Journal of Production Economics 27: 137-143.

Remy, J. (2004). "Resource constrained scheduling on multiple machines." Information Processing Letters 91(4): 177-182.

RezaAmin-Naseri, M. and MohammadAliBeheshti-Nia (2009). "Hybrid flow shop scheduling with parallel batching." International Journal of Production Economics 117: 185-196.

Salvador, M. S. (1973). A solution to a special case of flow shop scheduling problems. New York, Springer.

Santos, D. L., Hunsucker, J.L. (1995). "Global lower bounds for flowshop with multiple processors." European Journal of Operational Research 80: 112-120.

Santos, D. L., Hunsucker, J.L., J. L. Hunsucker, et al. (1996). "An evaluation of sequencing heuristics in flow shops with multiple processors." Computers and Industrial Engineering 30(4): 681-691.

Shao, H., H.-P. Chen., et al. (2008). Minimizing makespan for parallel batch processing machines with non-identical job sizes using a neural network approach. 2008 ICIEA 2008 3rd Conference on Industrial Electronics and Applications, Singapore.

Su, L.-H. (2003). "A hybrid two-stage flowshop with limited waiting time constraints." Computers and Industrial Engineering 44(3): 409 - 424

Sung, C. S. and Y. H. Kim (2002). "Minimizing makespan in a two-machine flowshop with dynamic arrivals allowed." Computers & Operations Research 29: 275-294.

Tang, L. and P. Liu (2009). "Minimizing makespan in a two-machine flowshop scheduling with batching and release time." Mathematical and Computer Modelling 2009: 1071-1077.

Vishwanathan, K., N. Kulkarni, et al. (2007). A Comparative Study of Dispatching Rules in Hybrid Flow Shops with Discrete and Batch Processors. Proceedings of the 2007 Industrial Engineering Research Conference.

Webster, S. T. (1996). "A general lower bound for the makespan problem." European Journal of Operational Research 89: 516-524.

Wittrock, R. J. (1985). "Scheduling algorithms for flexible flow lines." IBM Journal of Research and Development 29(401-412).

Wittrock, R. J. (1988). "An adaptable scheduling algorithm for flexible flow lines." Operation Research 36: 445-453.

Xu, S. and J. C. Bean (2007). A Genetic Algorithm for Scheduling Parallel Non-identical Batch Processing Machines. IEEE Symposium on Computational Intelligence in Scheduling.

Yang, Y. (1998). Optimization and Heuristic Algorithms for Flexible Flow shop Scheduling. Graduate School of Arts and Sciences, Columbia University. Ph.D.: 104.

# VITA

## YANMING ZHENG

| | |
|---|---|
| October 6, 1975 | Born, Xianyou, Fujian, China |
| 1994-1998 | B.A., Structure Engineering<br>Southeast University<br>Nanjing, Jiangsu, China |
| 1998-2001 | M.S., Construction Management<br>Southeast University<br>Nanjing, Jiangsu, China |
| 2001-2006 | Huawei Technologies, China |
| 2006-2010 | Teaching Assistant<br>Florida International University<br>Miami, Florida |
| 2009 | Doctoral Candidate in Industrial and Systems Engineering<br>Florida International University<br>Miami, Florida |

PUBLICATIONS

Yanming Zheng, Chin-Sheng Chen, and Syed M. Ahmed. (2007). *the Impact of MTO Feature on ERP Application in Construction Industry.* Proceedings of the Fourth International Conference on Construction in the 21st Century, Australia. 664-672.