AN ABSTRACT OF THE DISSERTATION OF

Omid Shahvari for the degree of Doctor of Philosophy in Industrial Engineering presented on July 24, 2017.

Title: Bi-Criteria Batching and Scheduling in Hybrid Flow Shops

Abstract approved: ____

Rasaratnam Logendran

In this research, a bi-criteria batching and scheduling problem is investigated in hybrid flow shop environments, where unrelated-parallel machines are run *simultaneously* with different capacities and eligibilities in processing, in some stages. The objective is to *simultaneously* minimize a linear combination of the total weighted completion time and total weighted tardiness. The first favors the producer's interest by minimizing work-in-process inventory, inventory holding cost, and energy consumption as well as maximizing machine utilization, while the second favors the customers' interest by maximizing customers' service level and delivery speed. In particular, it disregards the group technology assumptions (GTAs) by allowing for the possibility of splitting pre-determined groups of jobs into inconsistent batches in order to improve the operational efficiency. A comparison between the group scheduling and batch scheduling approaches reveals the outstanding performance of the batch scheduling approach. As a result, contrary to the GTAs, jobs belonging to a group might be processed on more than one machine as batches, but not all machines may be capable of processing all jobs. A sequence- and machine-dependent setup time is required between each of two consecutively scheduled batches belonging to different groups. Based on manufacturing company policy, the desired lower bounds on batch sizes are considered for the number of jobs assigned to batches. Although, the direction in which all jobs move through production line is the same, some jobs may skip some stages. Furthermore, to reflect real industry requirements, the job release

times and the machine availability times are considered to be dynamic, which means not all machines and jobs are available at the beginning of the planning horizon.

The problem is formulated with the help of four mixed-integer linear programming (MILP) models. Two out of four MILP models are formulated as two integrated phases, i.e., batching and scheduling phases, with respect to the precedence constraints between each pair of jobs/batches and/or the position concept within batches. The optimal combination between batch compositions of groups are determined in the batching phase, while the optimal assignment and sequence of batches on machines and sequence of jobs within batches are determined in the scheduling phase, with respect to a set of operational constraints. A batch composition of a group corresponding to a particular stage, determined in the batching phase of the MILP model, represents the number of batches assigned to the group as well as the number and type of jobs belonging to each batch of that group. Since the first and second MILP models lead to unmanageable solution space, the relaxed MILP model, which allocates one and only one job to each batch of each group in each stage, can be developed to focus on the non-dominated solution space. The optimal solutions of MILP models and relaxed MILP model are equal, if and only if the optimal solution of the relaxed MILP model does not violate the desired lower bounds on batch sizes. Since the relaxed MILP model cannot guarantee the optimal solution of the MILP models, a third MILP model is developed by integrating batching and scheduling phases. This MILP model eliminates an exhaustive combination enumeration between batch compositions of all groups in all stages. Although the third MILP model converges to the optimal solution slower than the relaxed MILP model, it guarantees finding the optimal solution of the first and second MILP models. A comparison between four MILP models shows the superior performance of the third MILP model.

However, since the problem is strongly NP-hard, it is not possible to find its optimal solution within a reasonable time as the problem size increases from small to medium to large, even by the relaxed MILP model or the fourth MILP model. Therefore, several meta-heuristic algorithms based upon basic local search, basic population-based search, and hybridization of local search and population-based searches are developed, which move back and forth between batching and scheduling phases. Tabu Search (TS) is implemented as a basic local search algorithm, while Tabu Search/Path-Relinking (TS/PR) is implemented as a local search algorithm enhanced with a population-based structure. TS is incorporated into the framework of path-relinking to exploit the information on good solutions. The TS/PR algorithm comprises several distinguishing features including relinking procedures to effectively explore trajectories connecting elite solutions and the methods used to choose the reference solution. Particle Swarm Optimization (PSO) is implemented as a basic population-based algorithm, while Particle Swarm Optimization enhanced with a local search algorithm (PSO/LSA) is developed to realize the benefits of batching and, consequently, enhance the quality of solutions.

Since there is interdependency between positions of a job in different stages of a hybrid flow shop in batch scheduling, a meta-heuristic algorithm is not capable of capturing these interdependencies and, subsequently, its efficacy can be diminished. In order to capture this interdependency, the non-, partial-complete-, and stage-based interdependency strategy are developed. In the stage-based-interdependency strategy, a complete sequence related to all of the stages is gradually determined, stage by stage. An initial solution finding mechanism is developed to trigger the search into the solution space and generate an initial population. The performances of these algorithms are compared to each other in order to identify which algorithm(s) outperforms the others. Nevertheless, the performances of the best algorithm(s) are evaluated with respect to a tight lower bound obtained from a branch-and-price (B&P) algorithm.

The B&P algorithm uses Dantzig-Wolfe decomposition (DWD) to divide the original problem into a master problem and several sub-problems (SPs) corresponding to each stage. The original problem is decomposed into the SPs by three DWDs corresponding to the three MILP models. Although, by applying DWD technique in the first and second MILP models, an exhaustive combination enumeration between batch compositions of all groups in all stages is excluded and, as a result, the SPs are easier to solve than the original problem, they are still strongly NP-hard because of an enormous number of combinations between batch compositions of all groups in each stage. However, the DWD technique corresponding to the relaxed MILP model not only drastically reduces the number of variables and constraints in the SPs, but also eliminates the batching phase of the first and second MILP models. Decomposing the original problem based on the relaxed MILP model and implementing the B&P algorithm cannot guarantee optimal solutions or tight lower bounds of problems unless the number of violations in the desired lower bounds on batch sizes is not significant. Therefore, the third MILP model is decomposed by DWD so that the B&P algorithm is capable of finding tight lower bounds even for large-size instances of the problem.

A comparison between the lower bounds obtained from the B&P algorithm and CPLEX reveals the impressive performance of the B&P algorithm, particularly for large-size problems. The evaluation of the best algorithms based upon these tight lower bounds developed by the B&P algorithm, uncovers the outstanding performance of hybrid algorithms compared to the results obtained from CPLEX.

Bi-Criteria Batching and Scheduling in Hybrid Flow Shops

by
Omid Shahvari

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented July 24, 2017
Commencement June 2018

Doctor of Philosophy dissertation of Omid Shahvari presented on July 24, 2017.

APPROVED:

Major Professor, representing Industrial Engineering

Head of the School of Mechanical, Industrial, and Manufacturing Engineering

Dean of the Graduate School

Omid Shahvari, Author

ACKNOWLEDGEMENTS

The completion of my dissertation and subsequent Ph.D. has been a long-standing desire to me for learning science and a great source of enthusiasm for developing new ideas. When I look back at all those moments I have engaged in this research, I see amazing changes not only in my life, but also in my character traits, all for the better, during this long journey. I owe these successes to several people.

I would like to express my sincere appreciation and thanks to my major professor, Dr. Rasaratnam Logendran. You have been a tremendous mentor and advisor for me. I want to thank you for encouraging and supporting my ideas. I appreciate all your help, persistence, enthusiasm, contributions, hard work, and continued guidance over the last four years. Thank you for patiently reading, correcting, and making suggestions for improvement on numerous papers and this dissertation. Special thanks for the time you spent to write letters of recommendation, which helped me a lot throughout my job search.

My gratitude extends to my committee members, Dr. Karl Haapala and Dr. Hector Vergara, for their guidance and support. Special thanks to Dr. Lisa Madsen for serving as my minor professor and for her useful feedback. I would like to thank Dr. Ellen Smit and Dr. Roger Graham for serving as the graduate council representative on my committee.

I would like to express my sincere appreciation to Dr. Nasser Salmasi for supporting me throughout my entire academic life. Thank you so much for your trust. I may not be where I am today without your support, guidance, encouragement, and advice. Thanks to Abbas Bozorgirad, Mohammad Yazdani, and Yasaman Mehravaran for their research as a great source of information. I would like to express my appreciation to the MIME staff members, Jean Robinson and Phyllis Helvie, for their help. Special thanks to Keith Price and Lori Burgeson who installed and maintained any software/hardware that I needed for my research.

My time at Oregon State University in Corvallis, Oregon, a beautiful and lovely small city, was a great chapter in my life due to the support and love of my friends. I will always be grateful to Abbas Abdolahi, Amir Javaheri, and Amin Mirkouei for their love and support and constantly cheering me up with their presence.

Last but not least, I would like to thank my family. Words cannot express how grateful I am to my father and my mother for their endless love, caring, dedication, and all the sacrifices they have made on my behalf over the years. Dad, you showed me how to be a human and love with no expectation. Mom, you taught me about hard work and persistence and were a great role model of resilience and strength. I am so proud of you and grateful for having you. To my sisters Manizheh, Mahnaz, and Mitra, I love you dearly with all my being and thank you for all of your advice, help, and caring. The last words of this acknowledgment go to my family: your love is the reason to live.

TABLE OF CONTENTS

Page

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

Bi-Criteria Batching and Scheduling in Hybrid Flow Shops

1. INTRODUCTION

Scheduling problems were first considered in the mid-1950s in small industries as minimizing the total flow time of a group of jobs on a single machine or minimizing the makespan of jobs on a couple of machines. Then, based on realistic requirements of different manufacturing industries, these problems got more complex over time. The intense competition faced by manufacturing companies since the late 1970s has made them very receptive to ideas that improve both operational effectiveness and competitive advantage. Foremost among these ideas that has achieved widespread acceptance is cellular manufacturing (CM), a widely-recognized production system, which combines the efficiency of flow shop production with the flexibility of job shop manufacturing.

With the advent of CM as a manufacturing philosophy, and as a result of scheduling of jobs that belong to the same family based upon their similarities in processing plans, tooling, shape and size, and setup, has become a topic of considerable interest among researchers and more suited for today's lot production systems (Li et al. 2010). Much of the attention in CM has focused on reduction in 1) total time required to setup the machines by implementing part family tooling and sequencing since cells process similar parts, 2) total flow times with the help of reducing setup times, move times, and waiting times, and 3) using small transfer batches. These provide the opportunity to reduce not only lot sizes, which leads to reduced work-in-process (WIP) inventories, but also manufacturing lead times and, subsequently market response times, which leads to improved customer satisfaction. Therefore, CM has focused on both producer's and customers' satisfaction.

CM is one of the applications of group technology, seeking to align process flows by families of component parts, where a portion of a firm's manufacturing system has been converted to cells. Group technology is a philosophy that capitalizes on product similarities and was designed as a means of improving manufacturing and design productivity in an era of rapidly expanding product diversity. A manufacturing cell is a cluster of dissimilar machines or processes located in close proximity and dedicated to the manufacture of families of parts. The parts belonging to each family are similar in their processing requirements including required operations, machine tool capacities, processing plans, tolerances, and so forth. The shop structure of each cell determines all characteristics of the environment where jobs are processed, such as the number and type of machines as well as the layout of the workshop. In order to decrease the cycle time of production, all machines are placed in serial stages based upon jobs' processing plans so that jobs should move through these stages in unidirectional passes. Also, in order to increase the flexibility of the production line, all machines related to bottleneck stages are placed in parallel, not necessarily all identical. This is a sophisticated flow shop known as hybrid flow shop.

Hybrid flow shop (HFS) is one of the advanced flow shop structures that have been studied in the scheduling literature. HFS is a flow shop, where at least one of the stages includes identical- and/or unrelated-parallel machines and jobs are required to be processed only on one machine. In contrast, a flow shop with only identical-parallel machines in at least one of the stages is called flexible flow shop (FFS). HFS environments have been utilized by many traditional industries including paper, textile, tobacco, pharmaceutical, metallurgical, oil, chemical, and food industry (Hsu et al. 2009, Zandieh et al. 2006). Semiconductor wafer fabrication, printed circuit board (PCB), and semiconductor light source manufacturing systems are modern electronics industries, which utilize HFS as a shop structure (Chang 2000, Choi et al. 2011, Jin et al. 2002, Neammanee and Reodecha 2009, Uzsoy et al. 1992, Wein 1988).

Effective scheduling approach is crucial to good performance of job shops and flow shops where a large variety of parts are produced. Therefore, a scheduling rule chosen for a manufacturing cell can have a strong impact on cell performance, especially when multiple incompatible job families are produced, virtual cellular manufacturing (VCM) is proposed, setup times are significant, and/or a manufacturing cell is operating at or near its capacity. Incompatible job families refer to multiple groups of dissimilar jobs, while the jobs assigned to a group are similar. Setup times need to be explicitly considered while scheduling decisions are made in order to increase productivity, eliminate waste, improve resource utilization, and meet deadlines (Allahverdi 2015). The research problem addressed in this paper focuses on determining the optimal schedule of incompatible job families (groups) in several consecutive cells of a manufacturing company, where there are negligible setup times (equal to zero) between two jobs belonging to the same group, while considerable setup times between two jobs of different groups. Therefore, the utilization of an appropriate scheduling approach along with an efficient optimization algorithm can have a significant effect in cell performance. There are three choices for scheduling of pre-determined groups of jobs based upon the industry requirements:

- *Group Scheduling (GS)*: jobs belonging to a group should be processed as a single batch, i.e., following the group technology assumptions (GTAs)
- *Batch Scheduling (BS)*: jobs belonging to a group might be processed as multiple batches, i.e., violating the GTAs but following the desired lower bounds on batch sizes, LB_b, (the minimum number of jobs assigned to batches, which are related to batch assignment on machines)
- *Job Scheduling (JS)*: jobs are processed irrespective of their assignment to groups, i.e., violating both the GTAs and LB_b.

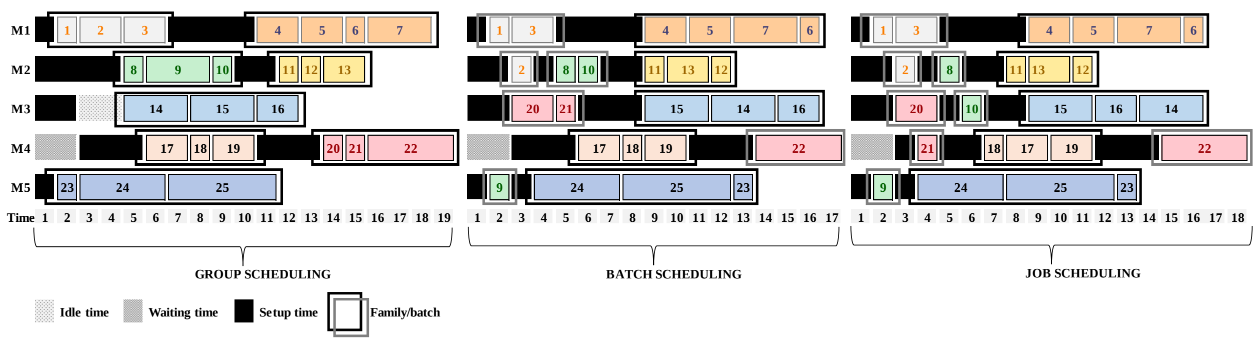


Figure 1. Illustration of job scheduling vs. batch scheduling vs. group scheduling

An illustration of GS, BS, and JS related to only one stage of HFS including five unrelated machines is depicted in Figure 1. By implementing BS instead of GS, each of 3 groups among the 8 groups are split into two batches. In addition, by implementing JS instead of GS, each of 2 groups among the 8 groups are split into three batches, while one group is split into two batches. In the following sub-section, an overview of semiconductor light sources manufacturing systems is presented in order to illustrate the application of the HFS environment. Apart from this, the contribution and motivation for considering other features of this research, such as violation in the GTAs, the desired lower bounds on batch sizes, and bi-criteria objective function are justified.

1.1. Industrial application

The semiconductor industry is the aggregate collection of companies engaged in the design and fabrication of semiconductor devices. Therefore, semiconductors are a primary input for nearly all electronic products. The semiconductor industry is widely recognized as a key driver for economic growth in its role as a multiple lever and technology enabler for the whole electronics value chain. Continuous growth but in a cyclical pattern with high volatility creates the need for high degrees of flexibility and innovation in semiconductors in order to constantly adjust to the rapid pace of change in the market. Panel production market is one of the fastest growing segments related to the semiconductor industry.

Market demand for panel products is rapidly changing and has experienced unexpected fluctuations. These variations, together with the extensive amount of customization requested from customers underlie the need for high degrees of flexibility and efficiency in this industry. The LCD, TFT-LCD, LED, LED-backlit LCD, OLED, and similar semiconductor light source manufacturing systems show rapid changes in semiconductor light source, and in general the panel production market. The above-mentioned manufacturing systems are considered as a middle stage in the electronics value chain, which receive semiconductors as their main input in order to produce LCDs, TFT-LCDs, LEDs, LED-backlit LCD, OLEDs, and other semiconductor light sources, which are themselves the main input for many other electronic industries. Revolutionary changes in types of equipment, materials, and production technologies have resulted in extraordinary advances in these industries. Most applications of semiconductor light sources are television sets, computer monitors, mobile phones, handheld video game systems, personal digital assistants, navigation systems, and projectors.

Different semiconductor light sources present different quality. The LCD, stands for Liquid Crystal Display, works by adjusting the amount of light blocked and usually has a backlight. The TFT-LCD, stands for Thin Film Transistor LCD, is a type of the LCD with a TFT attached to each pixel in order to improve the quality of images. The LED, stands for Light Emitting Diode, emits light when activated rather than blocking it like the LCD. The LED-backlit LCD uses LED backlighting instead of the cold cathode fluorescent (CCFL) backlighting used by most other LCDs. The OLED, stands for Organic LED, is comparatively recent technology in which the emissive electroluminescent layer is a film of organic compound that emits light in response to an electric current. Almost all panels that we currently use are TFT-LCDs, LED, or LED-backlit LCD. Figures 2 and 3 present a general form of components related to TFT-LCD and LED-backlit LCD panels, respectively. The components of the product are assembled on top of each other after they have been separately manufactured, either by the same company or by different suppliers.

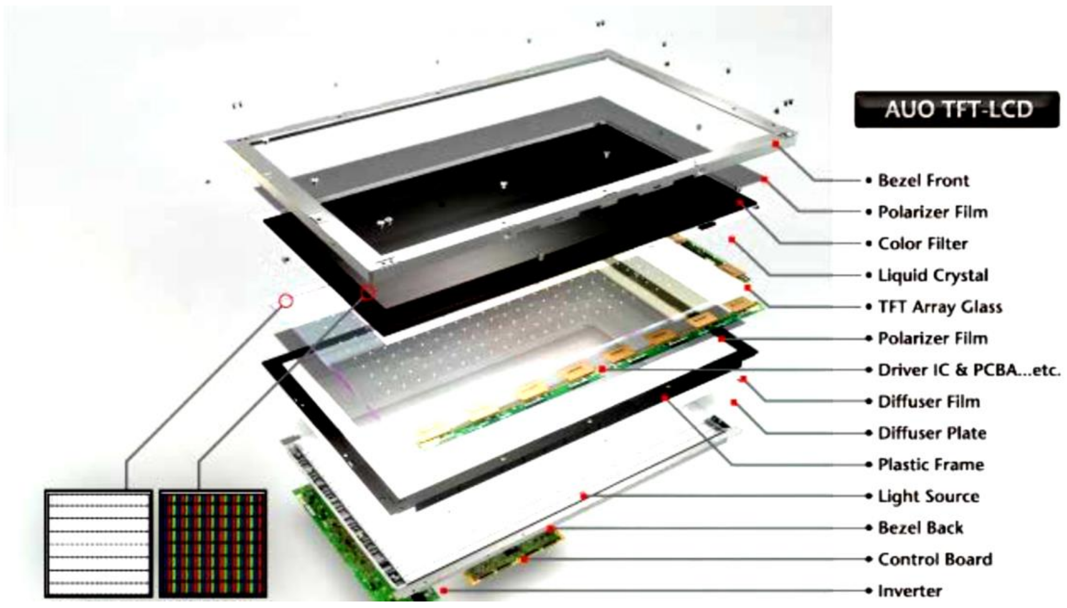


Figure 2. A general form of components in TFT-LCD panel

Each semiconductor light source has different types, which can be categorized based on manufacturing technology and other unique features. In addition to technological differences, each one is also categorized based on the size of the mother glass that is used in the early stages of the manufacturing process. Therefore, different groups of semiconductor light sources of the same type (for example LED-backlit LCD) are created in terms of shape, size, technological manufacturing, production planning, etc. Apart from this, each semiconductor light source manufacturing system includes a lot of individual operations, which can be divided into several main sub-processes such as fabrication process, cell process, and module process. Each sub-process performed in a cell of CM includes most complicated machines, which are typically automated or controlled by computers and placed in a serial sequence, i.e., flow shop. Several parallel machines are *simultaneously* run in the stages of a flow shop with long runtime (bottleneck stages) not only to decrease the cycle time of production, but also increase the flexibility of production at the same time. Parallel machines, which are expensive and have a wide range of specifications, include all types of identical, uniform and unrelated machines (Choi et al. 2011, Jeong et al. 2001, Shin and Leon 2004). Finally, the transportation of materials and maintenance of work-in-process (WIP) inventories between different stages of HFS are usually performed with the help of automated material handling systems (AMHS) such as automated guided vehicles (AGV), overhead shuttles (OHS), rail guided vehicles (RGV), and conveyer systems (Ho and Su 2010, Jang and Choi 2006).

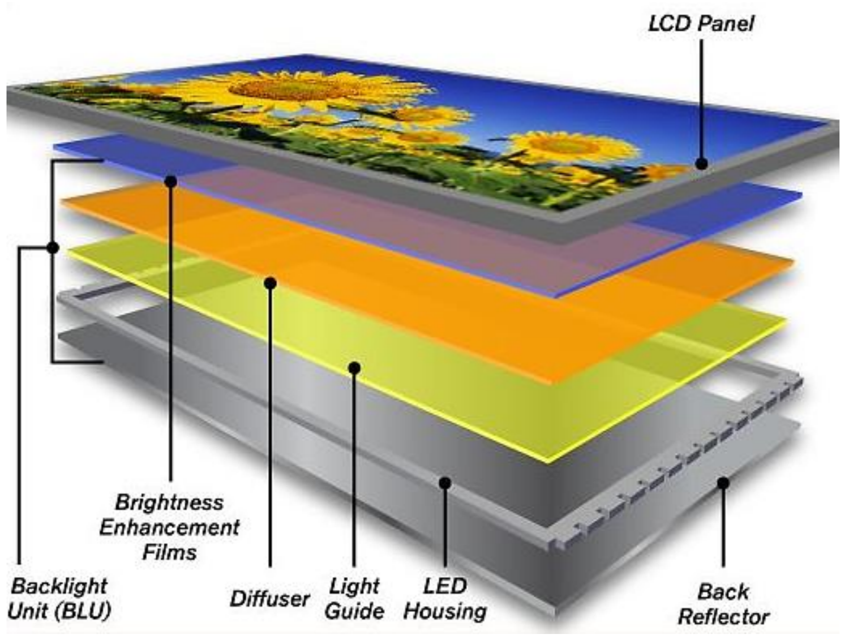


Figure 3. A general form of components in LED-backlit LCD panel

Last but not least, whenever the number of operations is increasing, effective scheduling approaches are necessary to keep not only high levels of productivity from producer's point of view, but also high levels of satisfaction from customers' point of view. Although scheduling techniques have been employed by the semiconductor and, subsequently, semiconductor light source manufacturing systems, these techniques need to evolve at the same step as other advances in these industries.

1.2. Motivation

This research specifically addresses scheduling of jobs clustered into pre-determined groups as inconsistent batches in an HFS environment, which is motivated by real industry applications. The mentioned batching and scheduling problem, simply known as batch scheduling, covers most of the challenges in the semiconductor light source manufacturing systems. Since, all jobs/parts should be processed as inconsistent batches in the same direction in an HFS environment, it lacks the flexibility of job shop production systems. However, as it has been described for the semiconductor light source manufacturing systems, efficiency and flexibility are two of the most crucial features of this industry, which is obtainable with the help of running parallel machines from different types in bottleneck stages. As a result, since these parallel machines have different rates in processing, a part has different run times on bottleneck stages.

The ever-changing design of products in the semiconductor light source manufacturing systems such as the LCD, TFT-LCD, LED, LED-backlit LCD, OLED, and others, together with huge seasonal demand for these products, and generally for related panels, make the introduction of flexibility within job shops in HFS production systems critical for practitioners. This novel production system is obtainable through CM, so that all parts are clustered into several families or groups in terms of their similarity and all required machines are also categorized into different cells in a way that each cell is almost capable of completely processing multiple groups of jobs. A few inter-cell movements are usually inevitable because complete disaggregation of cells is not always possible and it might be costly. Therefore, the combined flexibility and efficiency of this manufacturing structure makes it the best option for today's small-to-medium lot production systems (Li et al. 2010).

Instead of focusing only on optimizing the satisfaction of producers as in the literature on most scheduling problems, this research focuses also on customers' satisfaction. Therefore, the focus is on a bi-criteria objective of minimizing a linear combination of total weighted completion time and total weighted tardiness. From the producer's point of view, the objective is to not only minimize the total cost of work-in-process inventory, inventory holding cost, and energy consumption cost, but also to maximize machine utilization, both by minimizing the total flow time of all jobs. To attain the maximum customer satisfaction, the objective is to maximize customers' service level and delivery speed by minimizing the total tardiness for all jobs. These two criteria are combined with the help of a linear combination, which is referred to as the weighted sum technique in the literature.

Since the temperature of machines, for example in Polarizer Attachment process (one of the stages in semiconductor light source manufacturing systems including unrelated-parallel machines), is dependent on the size of cells (jobs), a set of cells might be processed contiguously as batches with regard to LB_b.

Although we assume the setup times between cells (jobs) belonging to the same batch is negligible (equal to zero), there is a slight difference in the machine temperature (tiny setup time) with regard to different cells (jobs) belonging to the same batch. Therefore, two cells (jobs) cannot be processed *simultaneously* by a machine because of different required temperatures. Since the jobs within different batches are not necessarily similar to each other, a sequence- and machine-dependent setup time is considered for switching processes from a batch of a group to a batch of different group on a particular machine. Furthermore, in order to depict the realistic requirements in industries, jobs are assumed to be released into the production systems at dynamic times, and machines are also assumed to be made available in the system at dynamic times. Also, some jobs/batches/groups can skip some stages because they do not need an operation to be performed in these stages.

As mentioned before, three scheduling approaches, used for sequencing part families as well as parts within each family in each cell of CM, are job scheduling, group scheduling, and batch scheduling. If the jobs assigned to a group belong to either one customer or different customers with different due dates, it might be better to process these jobs as multiple batches. Since GS follows the GTAs, it should process the jobs in one batch, while BS may choose to process the jobs in multiple batches. The benefits of integrating the batching decision into the GS approach are to reduce the completion time of jobs with the help of different machine capabilities and eligibilities, specifically in a bottleneck stage(s), perform timely processing of jobs with higher priority (based upon weight, tight due date, and earlier release time of jobs), and utilize the available machine capacities. In a bottleneck stage, since some machines are not eligible to process some jobs, a machine(s) might use its capacity either more than or less than other machines and, consequently, the completion time of a group(s) might be either lower than or higher than other groups when GS is applied. Contrary to GS, BS can potentially lead to a reduction in completion times of some jobs and/or their tardiness by processing the jobs with higher priorities. While this favors the producer's and/or customers' interests, it might change the production costs due to changes in completion time of other jobs and/or their tardiness as well as setup times (as multiple batches of the same group may have to be processed on different machines). So, there is a trade-off between changes in jobs' completion times and/or their tardiness on the one hand, and on the other, the changes in setup times. Although we do not expect to have small batch sizes with large setup times, the optimal schedule of JS may require the use of such batches. Therefore, there is no guarantee that better results would be obtained if the groups are allowed to divide into batches as much as possible, when the production costs are important. The manufacturing companies try to establish a balance between the setup time and the cumulative run time of each batch processed on a machine so that the production costs are not increased excessively. Thus, BS accompanied by LB_b is capable of identifying better solutions compared to GS and JS. Since group scheduling problems are well motivated by industrial applications, we implement batch scheduling instead of group scheduling in this research to enhance the efficiency and effectiveness of production systems.

A batch composition determines the number of batches assigned to a group as well as the number and the type of jobs assigned to each batch of that group, in a particular stage. The optimal batch composition of each group in each stage is determined in the batching phase of the problem. There is an enormous number of combinations between batch compositions of a group and, subsequently, an exhaustive combination enumeration between all batch compositions of all groups. Apart from this, the best assignment and sequence of batches on machines as well as job sequence within each batch should be determined for each enumeration in order to determine the optimal assignment and sequence of batches on machines and jobs within batches. Therefore, this feature as well as the previously noted features of the problem place it among the strongly NP-hard problems. As a result, it is not possible to find the optimal solutions for medium- and large-size problems, and even for small-size problems due to high complexity of the problem. The main approach in dealing with these problems is to develop a robust mathematical model and heuristic or meta-heuristic algorithms to find optimal or near optimal solutions. In addition, tight lower bounds are required to evaluate the performance of developed algorithms.

For illustration, the non-permutation group scheduling and batch scheduling in a three-stage hybrid flow shop are depicted in Figure 4. All groups have three jobs each, except one which has four jobs. The idle time and waiting time on a machine are due to the job release time and the machine availability time, respectively. By integrating the batching decision into GS, i.e., BS, the following results are obtained in the optimal schedule of BS:

- The third and fourth groups are divided into two batches, in each stage. Also, the first group is divided into two batches in the second stage.
    - A small batch including only job 3 with a reasonable setup time is created on the first machine of the second stage due to tight due date of this job.
    - Job 8 is processed as a small batch including only one job on the third machine of the second stage due to its higher priority and early release time.
- The completion time of all jobs and subsequently their tardiness are either reduced or not changed, except for two jobs (j_8 and j_9), which generally led to the reduction in total processing times.
- The job sequence within groups are changed in different stages due to either the job priority and/or the job release time, which blocks the job from starting early as a result of the split of some groups into batches.
- The run time of some inserted jobs on different machines are changed due to different capabilities of machines to process jobs.

- The available capacity created on the first machine of the third stage in GS (idle time before job 11) is utilized in BS, while there is idle time on the second machine of the second stage because the release time of job 4 had blocked this machine from processing this job earlier.
- And finally, the number of setups increases and some setup times change.

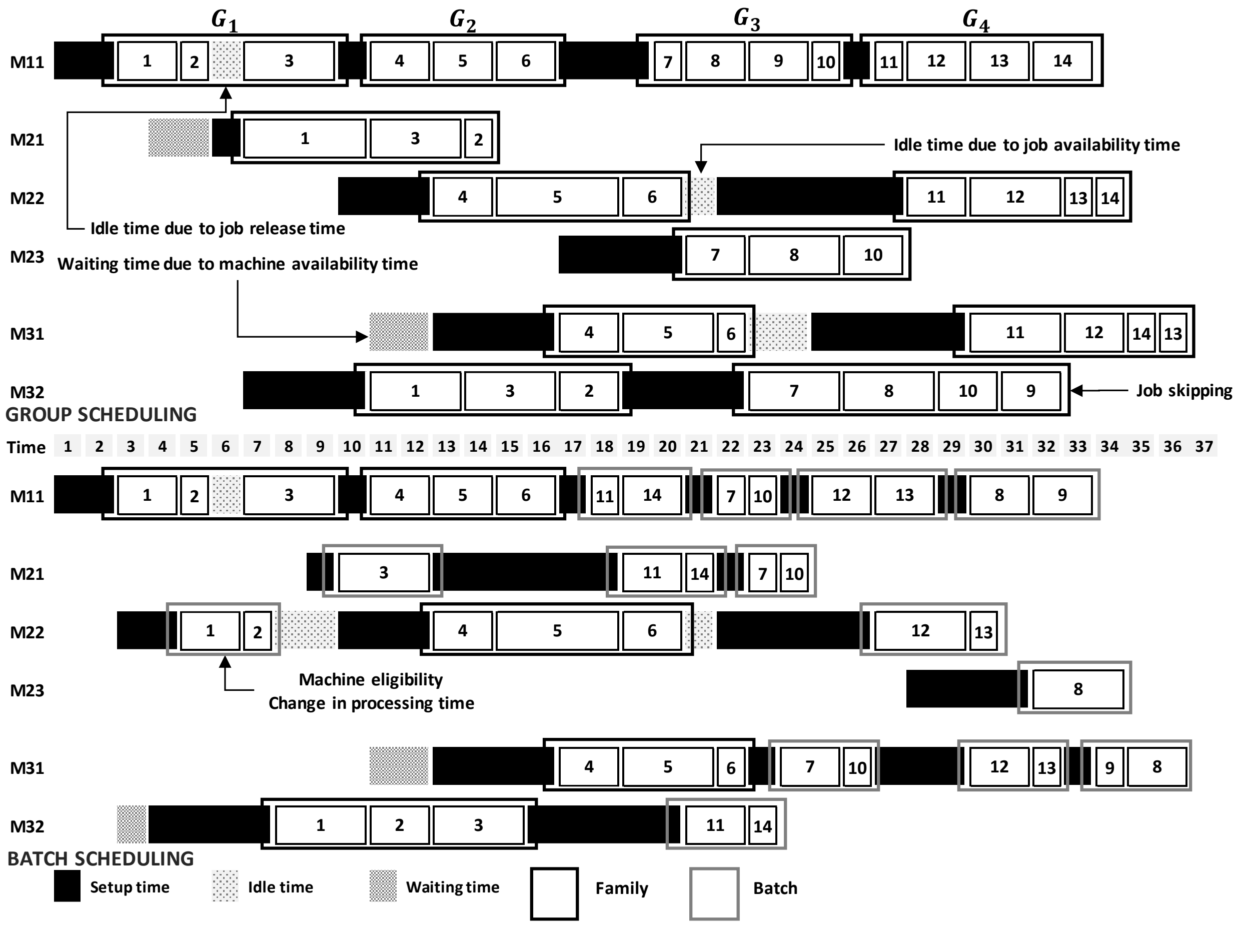


Figure 4. Illustration of batch scheduling vs. group scheduling

Therefore, it is clear that it might be advantageous to split one or more groups into batches, thus resulting in the reduction of job completion time and/or tardiness and, consequently, the reduction in the production cost. But doing so might increase completion time and/or tardiness of other job(s) and change in setup costs, as multiple batches of the same group may opt to be processed on different machines, thus resulting in the increase in the production cost. Therefore, the trade-off between the reduction and increase in the production cost can be used in favor of implementing BS by considering desired lower bounds.

1.3. Contributions

One of the main purposes of this research is to tackle the scheduling problem in such a manufacturing cell with the help of a mathematical model as well as providing a set of efficient meta-heuristic algorithms, specifically for industry-size problems. The orders for semiconductor light sources of the same group might be released to the manufacturing system by different customers with different due dates. Therefore, semiconductor light sources of the same group can be processed as multiple batches with respect to customers' priorities and batch development restrictions. Thus, another main purpose of the research is to show the benefits of integrating the batching decision in the traditional group scheduling approach, i.e., the novel batch scheduling approach. Despite the existing research on different types of hybrid flow shop scheduling problems, the lack of developing an efficient mathematical model and, consequently, an optimization algorithm for solving the research problem optimally in a reasonable computational time are recognizable in the literature. To the best of our knowledge, applying column generation in the framework of branch-and-bound algorithm (branch-and-price optimization algorithm) to find optimal solutions for large-size problems has not been implemented for the problem addressed in this research. The branch-and-price algorithm is efficient because of the use of flow conservation constraints.

Four mixed-integer linear programming (MILP) models are developed in terms of different concepts to mathematically represent the problem, optimally solve small-size instances of the problem, and construct a lower bounding mechanism in order to evaluate the performance of non-exact algorithms, particularly for large-size problems. Several meta-heuristic algorithms in terms of a local search structure, a population-based structure, and hybridization of local search and population-based structures are developed to find high quality solutions for the problem. These algorithms are developed based on the most effectively used meta-heuristics in the literature of HFS scheduling problems, i.e. tabu search (TS) as a local search algorithm, particle swarm optimization (PSO) as a population-based algorithm, and tabu search accompanied by path-relinking strategy (TS/PR) as well as particle swarm optimization accompanied by a local search algorithm (PSO/LSA) as hybrid meta-heuristic algorithms.

In addition, a lower bounding mechanism is constructed with the help of the column generation (CG) algorithm, also known as "pricing" problem. CG follows the Dantzig-Wolfe decomposition technique and decomposes the MILP model into a master problem (MS) and a sub-problem(s) (SP). Although the optimal solution of CG is guaranteed to be a lower bound for the original problem, this lower bound may not necessarily be one of high quality. Since CG relaxes the integer constraints of the master problem, a branching procedure (similar to the branch-and-bound (B&B) technique) is performed on fractional variables obtained from CG to construct a high quality lower bound, commonly referred to as a tight lower bound in the mathematical programming literature. As a result, collectively the mentioned approach is referred to as branch-and-price (B&P) technique. A graphical representation of the research contribution is provided in Figure 5.

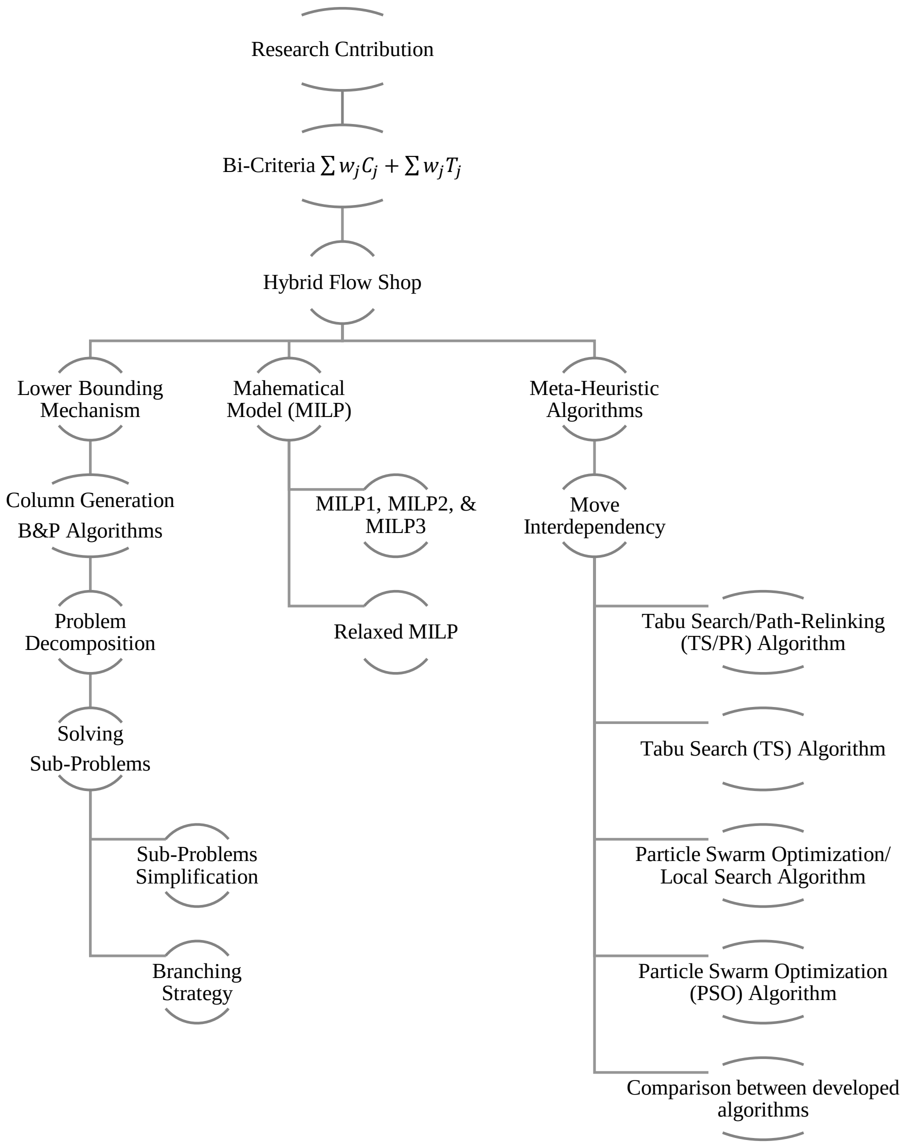

Research Cntribution
Bi-Criteria $\sum w_j C_j + \sum w_j T_j$
Hybrid Flow Shop
Lower Bounding Mechanism
Mahematical Model (MILP)
Meta-Heuristic Algorithms
Column Generation B&P Algorithms
MILP1, MILP2, & MILP3
Move Interdependency
Problem Decomposition
Relaxed MILP
Tabu Search/Path-Relinking (TS/PR) Algorithm
Solving Sub-Problems
Tabu Search (TS) Algorithm
Sub-Problems Simplification
Particle Swarm Optimization/ Local Search Algorithm
Branching Strategy
Particle Swarm Optimization (PSO) Algorithm
Comparison between developed algorithms

Figure 5. Research Contribution for the Problems Investigated

1.4. Research outline

Section 2 describes all features of the problem, in detail. These features include batching and scheduling phases incorporated into the batch scheduling problem, the objective function of the research problem, and the realistic situations and requirements in industries, i.e., sequence- and machine-dependent setup time, dynamic job release time, dynamic machine availability, machine capability and eligibility, stage skipping, and desired lower bounds on batch sizes.

Batch scheduling is motivated wherever group scheduling is applicable. Therefore, the literature review on both group scheduling and batch scheduling is provided on Section 3. Besides, the literature review on the main features of the research problem including hybrid flow shop environment, bi-criteria objective function, and meta-heuristic algorithms is provided in this section.

Section 4 explains the methodologies proposed to deal with the research problem. These methodologies include mathematical programing models, which can be used to optimally solve the problem, as well as several meta-heuristic algorithms that can find optimal or near optimal solutions. The common implementation strategies related to the search algorithms including move interdependency, initial solution/population, and refinement/adjustment step are explained. Subsequently, the main characteristics for each of the search algorithms are individually explained, in detail. The appropriate flow chart and/or pseudo-code is provided for each algorithm in this section.

The performance of developed meta-heuristics is evaluated with the help of a lower bounding mechanism, which is described in Section 5. A column generation technique is developed in this section. This section includes a brief background of this methodology, formulating master and sub problems and establishing valid lower bounds to the entire problem. Some properties on simplifying sub-problems to a problem with less complexity are also developed in this section.

A comprehensive data generation mechanism is described in Section 6 to develop test problems, followed by the experimental setup to evaluate the performance of developed algorithms and lower bounding mechanism in Section 7. Finally, Section 8 concludes the presentation of research.

2. PROBLEM STATEMENT

The problem addressed in this research is to schedule n different jobs clustered into g pre-determined groups as inconsistent batches, where each group contains n_i jobs (i.e., $\sum_{i=1}^{g} n_i = N$). The scheduling problems can be defined based on three-field notation $\alpha/\beta/\gamma$ developed by Graham et al. (1979). The first field (α) describes the shop (machine) setting. The second field (β) describes the setup information. Finally, the third field (γ) defines the performance measure. Therefore, the research problem addressed here is defined as $HF_m\left|ST_{sd,f}, r_j, a_j, M_j, LB_i, skip\right|F_l(\alpha \sum w_j C_j , \beta \sum w_j T_j)$. This problem includes the following features:

1. ***Batch scheduling:*** there are pre-determined groups (families) of jobs based on similarities in processing requirements. The GTAs restricts all jobs within a group to be processed successively and without interruption from other groups. Scheduling of groups of jobs in the presence of the GTAs is referred to as group scheduling. Splitting (batching) pre-determined groups of jobs into inconsistent batches together with scheduling developed batches on machines as well as jobs within batches (i.e., violating the GTAs by dividing groups into inconsistent batches), is called as batching and scheduling problem. In the literature, this is referred to simply as batch scheduling, wherein sub-groups belonging to each group of jobs are referred to as batches. It might be possible to split one or more groups into batches, so that all jobs of a batch are processed consecutively on the same machine. Realistically, a batch cannot be processed on a machine if there is at least one job in the batch, which cannot be processed on that machine. As a result, contrary to the GTAs, jobs belonging to a group might be processed on more than one machine as batches, but not all machines may be capable of processing all jobs. In each stage of batch scheduling, a decision needs to be made regarding both the batching phase and the scheduling phase.

1.1. ***Batching phase:*** a batch composition determines the number of batches assigned to a group as well as the number and the type of jobs assigned to each batch of that group with regards to desired lower bounds, in a particular stage. Therefore, the batching phase determines batch compositions of all groups for the entire stages. Since all jobs of each group can be processed as different batch compositions corresponding to all stages in batch scheduling, it is referred to as inconsistent batches.

1.2. ***Scheduling phase:*** the sequence of batches on machines as well as the sequence of jobs within batches are determined in the scheduling phase, for each combination of batch compositions of all groups for the entire stages, which was determined in the batching phase.

2. ***Hybrid flow shop:*** in order to decrease the cycle time and increase the flexibility of the production line, it follows the structure of a unidirectional hybrid flow shop, which includes unrelated-parallel machines (UPM) in at least one stage. Some parallel machines can also be identical. In HFS, the machines assigned to a stage with long runtimes (bottleneck stage) are run *simultaneously* with different capacities and eligibilities in processing.

3. ***Bi-criteria objective:*** the objective function composed of two criteria is to *simultaneously* minimize a linear combination of total weighted completion time and total weighted tardiness of jobs. The first favors the producer's interest by minimizing work-in-process (WIP) inventory, inventory holding cost, and energy consumption as well as maximizing machine utilization, while the second favors the customers' interest by maximizing customers' service level and delivery speed. These two criteria are combined with the help of normalized importance coefficients α and β, which are the weight attributed to the producer and customers, respectively ($0 < \alpha, \beta < 1$ and $\alpha + \beta = 1$). In order to capture the importance of different products from both the producer's and customers' view point, a weight is assigned to each job in the objective function.

4. ***Sequence- and machine-dependent setup time:*** processing a job by a machine requires the machine to be set up first. This setup takes a variable amount of time according to the previous configuration of the machine. Therefore, a setup is required between each of two consecutively scheduled batches belonging to different groups, which is dependent on both machine assignment and sequence of batches. Thus, the problem belongs to the class of sequence-dependent batch scheduling. Since all jobs within a batch are related to the same group and, subsequently, they are similar to each other, the setup time required to prepare a machine for switching the process from one job to another one is assumed to be negligible.

5. ***Dynamic job release time:*** the release times for jobs are considered to be dynamic, which means not all jobs are available at the beginning of the planning horizon (i.e., $t = 0$). Therefore, some jobs will be released at $t > 0$.

6. ***Dynamic machine availability time:*** the availability times of machines are also considered to be dynamic, which means not all machines are available at the beginning of the planning horizon (i.e., $t = 0$). Therefore, some machines will be available at $t > 0$. The dynamic machine availability time usually happens because of maintenance or operations being performed in the previous scheduling period.

Point: all job release times and machine availability times are deterministic and known before the scheduling process. Therefore, the problem is not among the stochastic scheduling problems.

7. ***Machine eligibility and capability:*** in HFS, the machines assigned to a bottleneck stage might not be eligible to process some jobs. This is referred to as machine eligibility, which is mainly because of technical incapability of machines. Apart from this, these machines might have different capacities in processing. Therefore, each job might have different run time in terms of machine assignment in bottleneck stages.

8. ***Desired lower bounds on batch sizes:*** there should be a balance between setup time and cumulative run time of each batch processed on a machine, which is determined by a manufacturing company's policy in terms of the minimum number of jobs assigned to a batch, i.e., the desired lower bounds on batch sizes.

   Point: although there is at least one machine to process consecutively all jobs of each group as a single batch in each stage, a batch cannot be processed on a machine if there is at least one job in the batch which cannot be processed on that machine and/or the number of jobs assigned to the batch is less than the desired lower bound on that machine.

9. ***Stage skipping:*** jobs should move through stages so that each job should be processed in at least one stage. Although, the direction in which all jobs move through is the same, meaning a flow-line arrangement as required by batch scheduling, some jobs and, subsequently some batches/groups may skip some stages because they do not need an operation to be performed in these stages.

3. LITERATURE REVIEW

Scheduling problems were first considered in the mid-1950s. The comprehensive reviews of scheduling problems from mid-1998 to mid-2006, reported by Allahverdi et al. (2008), document the advancement made in scheduling research over the years. Recent comprehensive review by Allahverdi (2015) including static, dynamic, deterministic, and stochastic environments, classify scheduling problems based upon shop environments as single machine, parallel machine, flow shop, job shop, or open shop, since the mid-2006 until the end of 2014. Generally, scheduling problems can be classified based upon a number of factors including setup time/cost, the number of stages jobs need to be processed, job processing requirements, the number of machines at each stage, and the performance measure to be optimized. The scope of this research is to improve the objective function value of group scheduling problems. Therefore, the literature reviews are presented on group scheduling problems with respect to different shop environment and implemented methodologies, along with some important characteristics of this research including the hybrid flow shop environment and bi-criteria objective function.

Group scheduling problems are studied by single-machine with different assumptions to capture the real industries' requirements. In today's competitive environments, some manufacturing industries insert additional machines to the single-machine shop environment to ensure attaining a specific quality, producing new products, and increasing both the flexibility and capacity of the production system. Shop structure is changed by inserting new machines in different positions of the single-machine shop environment, as parallel machines and flow shop, with respect to job processing plans.

When new machines are inserted in parallel, the parallel machine scheduling problems are created, which can be divided into three categories: Identical machines, P_m, where the run time of a job does not depend on the machine to which it is assigned; Uniform machines, Q_m, where machines have an associated speed for processing jobs at a consistent rate; and Unrelated machines, R_m, where each type of machines processes each job in different rates. So the run time of each job on unrelated-parallel machines depends on the machine to which it is assigned. Since unrelated-parallel machines (UPM) consider different run time for each job on each machine, it is more prevalent in industry than identical and uniform parallel machines (Allahverdi et al. 2008).

If the new machines are inserted in series in which the flow or movement of all jobs is the same, from the first machine in the series to the last one, flow shop (FS) structure is developed. The sophisticated shop structures are flexible flow shop (FFS) and hybrid flow shop (HFS), when at least one of the serial stages of a unidirectional flow shop includes identical- and unrelated-parallel machines, respectively. These extended layouts are sometimes addressed as flow shop with multiple machines, multiprocessor flow shop, or flow shop with parallel machines. The other reason for adding some machines to a stage of a flow shop is to balance the capacity of the flow shop and increase in demand for customized products.

3.1. Review of the literature related to hybrid flow shop environments

The HFS scheduling problem might be considered as a generalization of two particular types of scheduling problems: the UPM scheduling problem and the FS scheduling problem. The allocation of jobs to machines and the sequence of jobs through the shop are the key decisions of the UPM and FS, respectively. Hence, once the configuration of HFS has been designed, the main decisions in the operation of HFS are to assign and to schedule the jobs to the machines in each stage according to one or several given criteria. Since, HFS in this research has batching constraints and bi-criteria objective function, the scheduling approach has a large impact on the performance of HFS. On the other hand, although HFS increases the productivity and flexibility of production, it directly has an impact on the complexity of the scheduling problem. Therefore, HFS is one of the most important characteristic of this research.

A comprehensive set of papers dealing with HFS problems is studied by Ruiz and Vázquez-Rodríguez (2010). The earliest work on HFS problems started with considering identical-parallel machines in flow shop so that jobs are not allowed to skip any stage. The simplified versions considered two or three stages including one or two machines in each stage. The exact algorithms such as B&B and dynamic programming (DP) have been developed for these problems with unlimited number of stages and machines (Brah and Hunsucker 1991, Rajendran and Chaudhuri 1992) and more generalized versions of these problems have been implicitly solved by developing mathematical programming models (Liu and Karimi 2008, Tang and Xuan 2006).

Ruiz and Maroto (2006) developed hybrid algorithms based on GA-based algorithms with different local search algorithms to address an HFS scheduling problem with consideration of sequence-dependent setup times and machine eligibilities. Chen and Chen (2009) developed bottleneck-based heuristic algorithms to find the minimum total tardiness in an HFS scheduling problem. The comparison between these algorithms against several commonly used dispatching rules as well as a TS-based algorithms revealed that bottleneck-based algorithms are not capable of finding solutions better than TS. Jungwattanakit et al. (2009) studied an HFS problem with the purpose of minimizing a linear combination of the makespan and the mean tardiness of all jobs. They considered sequence-dependent setup times and dynamic job release times. In addition to developing a mixed-integer programming (MIP) model, they developed and compared the performance of different algorithms based on TS, SA and GA.

Yaurima et al. (2009) proposed some algorithms based on GA to find the minimum makespan in an HFS scheduling problem. The underlying assumptions for this problem were sequence-dependent setup times, machine eligibility, and limited buffers. In addition to the mentioned assumptions, Ruiz et al. (2008) considered a set of comprehensive assumptions such as stage skipping and job release time for the same problem. The researchers developed an MIP model as well as several heuristics to deal with the problem. Zandieh and Karimi (2011) developed an adaptive multi-population GA to solve the multi-objective group scheduling problem in an HFS environment with sequence-dependent setup times. The objective was to minimize the summation of makespan and total weighted tardiness in a group scheduling problem. The underlying assumption for their problem was that the jobs were allowed to skip some stages.

Despite the comprehensive research that has been done on different varieties of HFS scheduling problems, two important gaps are recognizable in the literature for these problems. The first gap is the lack of violation in the GTAs in any of the HFS scheduling problems. To the best of our knowledge, the HFS scheduling problems have been studied so far by considering the GTAs (particularly the ones that follow the cellular manufacturing concepts). The second gap is the lack of consideration of learning effects in the HFS scheduling problems. Ignoring learning effects, while scheduling jobs, may result in sub-optimal solutions.

3.2. Review of the literature related to group scheduling

Group technology is a philosophy in which similar parts are clustered into different families in order to take advantage of the similarities in both design and production. CM is one of the applications of group technology, seeking to align process flows by families of component parts, where a portion of a firm's manufacturing system has been converted to cells. Therefore, CM along with group technology represent a flexible manufacturing system in which the sequence of families/groups on machines as well as the sequence of parts/jobs within groups in each cell, can be determined with the help of group scheduling with respect to the GTAs.

Single machine

The study of group scheduling problems was initiated with the simplest shop structure, i.e. single-machine problems with different assumptions and constraints, such as single or multi-criteria objective function, independent or dependent setup times, and many other assumptions to reveal the real-world scheduling problems (Cheng et al. 1999, Sun et al. 1999, Wang et al. 1999, Webster and Baker 1995). A sequence-independent group scheduling problem on a single-machine is studied by Li et al. (2011) with respect to several objective functions including earliness, tardiness, due date assignment, and flow time costs. Gupta and Chantaravarapan (2008) presented an MILP model for a single-machine group scheduling with family setups to minimize total tardiness. They also proposed two-phase heuristic algorithms, including SA. Empirical results indicated that the heuristics are effective. Recent researches in a single-machine group scheduling dealt more with deterioration and learning effect assumptions. Bai et al. (2012) studied a single-machine group scheduling problem with general deterioration and learning effect. They showed that the problem is polynomially solvable. Liu et al. (2014) considered the group scheduling on a single-machine with deteriorating setup and processing times where both setup and processing times are increasing functions of their starting times. Their primary objective is to minimize total weighted completion time, while the secondary objective is to minimize maximum cost. They presented a polynomial time algorithm. Along with learning effect and deterioration assumptions, single-machine group scheduling with other assumptions including time dependent processing times, ready times, allotted resource, and past-sequence-dependent setup times have been studied (Low and Lin 2012, Wang et al. 2014, Wang and Wang 2014). Yazdani Sabouni and Logendran (2013) considered the problem of minimizing the makespan on a single machine with carryover sequence-dependent setup times in PCB manufacturing.

Parallel machines

Bozorgirad and Logendran (2012) studied sequence-dependent group scheduling problem on unrelated-parallel machine with respect to a bi-criteria objective function, which simultaneously minimizes total completion time and total tardiness. Behnamian et al. (2010) addressed sequence-dependent group scheduling on a set of identical-parallel machines with due windows for jobs, i.e., each job has an interval rather than a single value. They proposed a multi-phase covering Pareto-optimal front method by using a multi-phase algorithm iterating over a GA in the first phase and three hybrid metaheuristics in the second and third phases. They showed that the multi-phase method is a better tool to approximate the efficient set than the global archive sub-population GA presented previously.

Flow shop

Three lower bounds were developed by Liou and Liu (2010) for sequence-dependent two-machine flow shop group scheduling problems. They also presented a PSO algorithm and evaluated its performance with the developed lower bounds. Liou et al. (2013) addressed a new encoding scheme-based hybrid algorithm for minimizing two-machine flow shop group scheduling problem with transportation times and sequence-dependent family removal times. They presented some lower bounds and proposed a hybrid heuristic consisting of PSO and GA. Logendran et al. (2006b) developed different TS-based algorithms to minimize the total completion time for a two-machine flow shop group scheduling problem. Salmasi et al. (2010) studied the total flow time minimization for a sequence-dependent group scheduling problem in a flow shop environment. They developed a mathematical programming model for small size problems, while a TS-based algorithm as well as a hybrid ant colony optimization (HACO) algorithm are developed to deal with large size problems. The performances of these algorithms have also been evaluated against a tight lower bound obtained from a B&P approach. Then, Salmasi et al. (2011) proposed a mathematical programming model as well as a hybrid ACO algorithm for a flow shop sequence-dependent group scheduling problem to minimize the makespan. Gelogullari and Logendran (2010) studied a carry-over sequence-dependent group scheduling problem in a flow shop environment for the assembly of PCBs in electronic manufacturing systems. They developed several TS-based algorithms to find the best sequence of groups as well as jobs. The performance of these algorithms was also evaluated with the help of a B&P algorithm. A fast hybrid PSO algorithm (Hajinejad et al. 2011) and efficient upper and lower bounding methods (Keshavarz and Salmasi 2014) were developed for a flow shop sequence-dependent group scheduling problem.

Flexible flow shop

Logendran et al. (2005) developed heuristic algorithms to minimize the makespan of a sequence-independent group scheduling problem in FFS environments. A similar study has been conducted by Logendran et al. (2006a) to minimize the makespan of a sequence-dependent group scheduling problem in an FFS environment with the help of TS-based algorithms. This work has also been continued by Shahvari et al. (2012) to develop a mathematical programming model for the problem as well as efficient TS-based algorithms to find the optimal or near optimal solutions. Keshavarz and Salmasi (2013) developed an MILP model for sequence-dependent group scheduling problem in an FFS environment and presented a memetic algorithm (MA). They also proposed a lower bounding technique and showed that their MA outperforms the TS algorithm of Shahvari et al. (2012). Luo et al. (2012) considered the GTAs with inconsistent family formation to minimize the makespan of the sequence in an FFS environment.

3.3. Review of the literature related to batch scheduling

Reviews of batch scheduling problems include those by Potts and Van Wassenhove (1992), Webster and Baker (1995), and Potts and Kovalyov (2000). Most of the batch scheduling problems consider either processing batches by batching machines or processing parallel batches by multiple parallel machines. In both cases, a set of jobs processed *simultaneously* as batches are completed together as long as the machine capacity is not exceeded. These batch scheduling problems determine only the sequence of batches, irrespective of the job sequence within each batch since the jobs assigned to each batch are processed simultaneously. The problem addressed in this research focuses on a hybrid flow shop batch scheduling problem, wherein scheduling of jobs that belong to pre-determined groups is permissible by splitting them into multiple batches, but the jobs in a batch formed from a group must be processed consecutively on a machine. Therefore, batch scheduling investigated in this research clearly is contrastingly different from the previous batch scheduling problems, because it assumes processing of a batch is completed when consecutive processing of all jobs within the batch (instead of *simultaneous* processing) is finished.

The study of batch scheduling problems (i.e., group scheduling without the GTAs) was initiated by a non-permutation flow shop batching and scheduling problem with sequence-dependent setup time by minimizing makespan Shen et al. (2014). They developed a tabu search heuristic, which contains several neighborhood functions, double tabu list, and a multilevel diversification structure. Shahvari and Logendran (2015) performed a preliminary investigation of a batching and scheduling problem on unrelated-parallel machines with respect to a bi-criteria objective function, which simultaneously minimizes a linear combination of total weighted completion time and total weighted tardiness. They developed a TS-based heuristic, which contains three levels with specialized tabu list for each level. The applicability of developed TS-based algorithm was demonstrated with the help of an example problem. This study was extended to cover a variety of research issues and insightful findings (Shahvari and Logendran 2017). These include, but not limited to, addressing the relative performance of batch scheduling by considering a benchmark of group scheduling problems on unrelated-parallel machines with the same bi-criteria objective function, developing a mathematical programming model to evaluate the performance of the TS-based algorithms with the help of a detailed statistical experimental design, and, more importantly, identifying ineffective neighborhoods in the implementation of the TS-based algorithms by developing and proving several theoretical properties with the help of lemmas. Later, Shahvari and Logendran (2016a) extended their work from a single stage problem to a multiple stages problem, i.e., hybrid flow shop. They addressed the hybrid flow shop batching and scheduling problem with the same bi-criteria objective function where sequence-dependent family setup times are present. A benchmark of small size problems is considered to show the superior performance of batch scheduling compared to group scheduling (Shahvari and Logendran 2016b). They developed two algorithms, which incorporated tabu search into the framework of path-relinking to exploit the information on good solutions. These tabu search/path-relinking algorithms comprised several distinguishing features including two relinking procedures to effectively construct paths and the stage-based improvement procedure to consider the move interdependency. In all mentioned works related to Shahvari and Logendran (2015, 2016a, 2016b, 2017), the efficiency and effectiveness of the proposed search algorithms were verified by comparing the results of these algorithms with optimal solutions obtained from CPLEX for small size problems. Apart from this, a wide range of realistic characteristics such as sequence-dependent family setup times, dynamic job releases, dynamic machine availability, machine eligibility and stage skipping (for HFS) was considered with the help of data generation mechanism. Also, the initial solution finding mechanism was implemented to trigger the search into the solution space.

3.4. Review of the literature related to bi-criteria scheduling problems

The objective function is one of the challenges in dealing with scheduling problems. In most of the literature for scheduling problems, the focus has only been on optimizing the satisfaction of producers. Minimizing the completion time is desirable for the producer so they can minimize their work-in-process (WIP) inventory as well as production costs; however, Armentano and Ronconi (1999) recognized that lots of manufacturers are now more interested in meeting the customers' due dates and maximizing the customers' service level by minimizing the tardiness. The use of a bi-criteria objective is motivated by the fact that successful companies in today's environment not only try to minimize their own cost but also try to fulfill their customers' need. Successful companies are those that consider both producer's and customers' needs. Therefore, a bi-criteria objective function can truly achieve the real-world stipulation more than a single criterion. Trying to optimize two mentioned objectives enables incorporating the coordination that must be maintained between the producer and the customers in scheduling problems. The satisfaction of both completion time and tardiness objectives moves in the same direction and hence the objectives are not in conflict. In other words, tardiness of a job is either reduced or not changed when its completion time is reduced. Comprehensive reviews of the literature on multi-objective scheduling problems include Behnamian et al. (2011), Dugardin et al. (2010), Mehravaran and Logendran (2011), Rana and Singh (1994), Tavakkoli-Moghaddam et al. (2010).

There are a couple of studies corresponding to *simultaneously* minimizing total completion time and total tardiness. Bi-criteria scheduling problem with sequence-dependent setup times on a single machine is considered by Eren and Güner (2006). The objective function of the problem was minimization of the weighted sum of total completion time and total tardiness. They proposed an effective mixed-integer programming model to find the optimum schedule for problems with up to 12 jobs, while for solving problems containing large number of jobs a special heuristic algorithm based upon tabu search was proposed.

Mehravaran and Logendran (2011) considered an unrelated-parallel machine job scheduling problem with sequence-dependent setup times to jointly minimize the work-in-process inventory for the producer and to maximize the customers' service level in a supply chain. Later, Mehravaran and Logendran (2012) studied a flow shop scheduling problem with sequence-dependent setup times and the same bi-criteria objective function. They considered permutation and non-permutation schedules in finding the optimal schedule for a flow shop as well as the operational constraints commonly encountered in the industry, including dynamic machine availabilities, dynamic job releases, and jobs skipping. In both mentioned works, they assessed the effectiveness and efficiency of the search algorithm by comparing the search algorithm solutions with that of the optimal solutions obtained from CPLEX in solvable small problem instances.

Xu and Yin (2011) proposed a corrected integer programming model that was proposed by Eren and Güner (2006) for a flow shop scheduling problem that was incorrect. They both considered the same bi-criteria objective function including a linear combination of total completion time and total tardiness. Ribas-Vila et al. (2009) presented and evaluated six simple heuristic algorithms for the problem of sequence-dependent identical-parallel machines with respect to simultaneously minimizing total completion time and total tardiness.

A bi-criteria group scheduling problem in a flow shop with sequence-dependent setup time was investigated by Lu and Logendran (2013) with dynamic job releases and machine availabilities. The goal was to minimize the weighted sum of total weighted completion time and total weighted tardiness. A mathematical model was also developed and implemented to evaluate the optimality of the results from search algorithms based on tabu search for small size problems.

Bozorgirad and Logendran (2012) addressed a sequence-dependent group scheduling problem on a set of unrelated-parallel machines. Later, Bozorgirad and Logendran (2013) extended their work to address a sequence-dependent group scheduling problem in hybrid flow shop where the parallel machines in one or more stages of the flow shop are unrelated and have different run times for the same job. Similar to previous work, the objective of the problem was to simultaneously decrease the producer's cost by minimizing the WIP and increase the customers' satisfaction by minimizing the total tardiness. In both mentioned works, the efficiency and effectiveness of the proposed search algorithms were verified by comparing the results of these algorithms with optimal solutions obtained from CPLEX for small size problems. They assumed that all of the jobs and machines may not be ready at time zero, meaning that they can be released at different times during the scheduling period. Apart from this, job skipping and group skipping were assumed for the second work. Bozorgirad and Logendran (2014) also proposed a lower bounding mechanism for the previous work without considering machine availability times. They proposed a lower bounding mechanism, based on the column generation algorithm that is able to find lower bounds, which are remarkably tighter than the bounds from CPLEX.

Shahvari and Logendran (2015, 2016a, 2016b, 2017) addressed a bi-criteria batch scheduling problem on unrelated-parallel machines and hybrid flow shop environments where the GTAs were disregarded. The mentioned works are completely reviewed in the literature related to batch scheduling.

3.5. Review of the literature related to the methodologies

Ruiz and Vázquez-Rodríguez (2010) classified all techniques in dealing with a variety of scheduling problems in HFS environments into three broad categories, i.e., exact algorithms, deterministic heuristics, and meta-heuristics. The exact algorithms, B&B, and DP are the most preferred algorithms for optimally solving very simplified versions of HFS scheduling problems, i.e., problems with a limited number of stages (mostly two or three stages) and machines in each stage (usually one or two machines in each stage) (Dessouky et al. 1998, Gupta et al. 1997, Haouari et al. 2006).

The complexity of most of HFS scheduling problems on one hand along with considering a wide variety of processing constraints, production requirements, resource and precedency constraints, buffer limits, machine eligibility and different types of objective functions on the other hand, lead to developing mathematical programming models for optimally solving the complicated HFS scheduling problems (Liu and Karimi 2008, Sawik 2000, Tang and Xuan 2006). Although B&B algorithms have also been used to solve complicated versions of HFS scheduling problems, mathematical programming models are developed to present mathematically sophisticated scheduling constraints and solve small-size problems. In addition, the mathematical programming models have been used as a basis to implement the B&B algorithm.

Since most of HFS scheduling problems are among the strongly NP-hard problems and their developed mathematical programming models should implement different optimization solvers based on exact algorithms, such as B&B or DP, these exact methodologies will not be able to optimally solve the problem within a polynomial/reasonable time. Therefore, deterministic heuristics and meta-heuristic algorithms are commonly used methodologies to deal with complicated HFS scheduling problems. The performance of deterministic heuristics, known as tailored heuristics or dispatching rules, is highly dependent on the structure of the problem and is usually lower than meta-heuristic algorithms. The higher performance in meta-heuristics is due to the avoidance of getting trapped into local optimal solutions. Based on a comprehensive survey on the literature related to methodologies implemented for HFS scheduling problems (Ruiz and Vázquez-Rodríguez 2010), TS, SA and GA are three of the most commonly used meta-heuristics in dealing with the complicated HFS scheduling problems. The permutation assumption cannot be followed by batch scheduling problems investigated in this research due to different batch compositions of all groups in all stages. Thus, it seems a local search algorithm enhanced with a population-based structure or a population-based algorithm enhanced with a local search structure will have a better performance compared to a basic local search algorithm and a population-based algorithm, respectively. Therefore, four types of the most commonly used meta-heuristics, i.e., TS as a basic local search algorithm and TS/PR as a local search algorithm enhanced with a population-based structure, PSO as a basic population-based algorithm and PSO enhanced with a local search structure, are proposed in order to capture the move interdependency between stages. A brief of the literature corresponding to these algorithms is presented next.

3.5.1. Review of the literature related to Tabu Search

TS, introduced by Glover (1986), is a remarkably successful algorithm for solving hard combinatorial problems. The application of TS on a large number of combinatorial optimization problems has shown its effectiveness in solving this type of problem (Logendran and Sonthinen 1997). The application of TS started with solving simplified versions of single-machine problems (Laguna et al. 1991). Then, TS was developed for single-machine problems with sophisticated scheduling constraint and assumptions.

Due to the characteristics of TS, such as its ability to avoid getting trapped in a local optima and identify multiple-local optima in exploring the solution space, it was a promising technique for solving scheduling problems with more realistic constraints and assumptions, in different shop environments. With respect to parallel machines environments, TS showed superior performance compared to other existing solutions (Kang et al. 2007, Kim et al. 2006, Lee et al. 2013). Logendran and Subur (2004) and Logendran et al. (2007) implemented TS to minimize the total weighed tardiness of unrelated-parallel machine scheduling problems with consideration of job splitting and sequence-dependent setup times, respectively.

TS has also been shown to be very effective in dealing with flow shop scheduling problems, although most of these problems were restricted to permutation sequences (Armentano and Ronconi 1999, Ben-Daya and Al-Fawzan 1998, Nowicki and Smutnicki 1996). Hendizadeh et al. (2008) proposed meta-heuristic algorithms based on TS by applying the concept of elitism and the acceptance of worse move to improve the intensification and diversification of moves for sequence-dependent flow shop with respect to minimizing the makespan. Salmasi et al. (2011) developed a TS-based algorithm to minimize the makespan of a sequence-dependent group scheduling problem in a flow shop environment. Shahvari et al. (2012) developed six meta-heuristic algorithms based on TS to minimize the makespan of a sequence-dependent group scheduling problem in an FFS environment. Bozorgirad and Logendran (2013) addressed sequence-dependent group scheduling in a hybrid flow shop problem with bi-criteria objective function and presented an MILP model and proposed four TS algorithms. They showed that one of the TS algorithms performs well.

TS was successfully implemented for scheduling problems with bi-criteria and multi-criteria objective function and sequence-dependent setup times (Bozorgirad and Logendran 2012, 2013, 2014, Choobineh et al. 2006). Eren (2007) developed an integer programming model and presented heuristics based on TS and random search for two-stage flow shop with multi-criteria objective function including total completion time, makespan, maximum tardiness and maximum earliness. He showed that the heuristic based on TS performs better than those based on a random search.

3.5.2. Review of the literature related to Tabu Search/Path-Relinking

A hybridization of some algorithms with TS-based algorithm leads to improving the performance of basic TS. Pacheco et al. (2013) proposed a heuristic method, hybridization of multi-start strategies with TS, for sequence-dependent single-machine problem with makespan minimization and they showed that their hybridized heuristic outperforms a metaheuristic based on GRASP. Allahverdi and Al-Anzi (2009) proposed three heuristics for assembly flow shop with minimization of total completion time. The three heuristics were a hybrid TS and two versions of self-adaptive differential evolution algorithm. They showed that one version of the self-adaptive differential evolution algorithm performs much better than the other version and the hybrid TS. Varmazyar and Salmasi (2012) presented an MILP model and proposed several metaheuristics based on TS and Imperialist Competitive Algorithm (ICA). They showed that the hybrid heuristic of TS and ICA performs the best. The sequence-dependent family setup time in a flow shop scheduling problem with minimization of total flow time was addressed by Salmasi et al. (2010) for the first time. They proposed a mathematical programming model and a branch-and-price algorithm. In addition, they presented a TS algorithm along with a hybrid ACO algorithm. The computational analysis indicated that the hybrid approach performs better than TS.

Due to the lack of mechanisms that exploit the information on good solutions, the performance of a straightforward TS might be unsatisfactory, even when it is accompanied by delicate memory structures and effective neighborhood mechanisms (Jia and Hu 2014). Thus, an auxiliary heuristic, namely path-relinking (PR), is incorporated into the basic TS-based algorithm. PR is an enhancement to TS-based procedure, leading to significant improvements in the solution quality. In principle, the tabu search/path-relinking (TS/PR) algorithm repeatedly operates back and forth between a path relinking method that is used to generate promising solutions on the trajectory set up from an initiating solution to a guiding solution, and a TS procedure that improves the generated promising solution to a local optimum (Peng et al. 2015).

PR is intimately related to the TS-based meta-heuristic and derive additional advantages with the help of adaptive memory and associated memory-exploiting mechanisms that are capable of being adapted to particular contexts, such as job shop and flexible job shop scheduling problems (Jia and Hu 2014, Peng et al. 2015). Due to the lack of exploiting the information on good solutions in the HFS scheduling problems, Shahvari and Logendran (2016b) developed a local search algorithm enhanced with a population-based structure, which is called TS/PR algorithm and comprises several distinguishing features such as two relinking procedures to effectively construct paths. The results showed that the performance of TS/PR is better than basic TS for batch scheduling in HFS. The reason is that the performance of basic TS is diminished due to different batch compositions of groups in different stages. Therefore, PR is incorporated into a basic TS in order to increase its performance by exploring on the information of good quality solutions in the solution space.

3.5.3. Review of the literature related to Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a fast-evolutionary algorithm, which is applied on a population of candidate solutions (Eberhart and Kennedy 1995). PSO has gained much attention in a variety of fields, particularly for continuous optimization problems. Based on a few scheduling problems in flow shop and flexible flow shop environments, the encouraging performance of the PSO algorithm in scheduling problems, and the increased application of the PSO algorithm in different scheduling areas, we are motivated to apply the PSO algorithm for the research problem.

Research reported in the past shows a significant interest in implementing the PSO algorithm for solving scheduling problems. Tasgetiren et al. (2004b) develop a PSO algorithm for the single machine total weighted tardiness scheduling problem. They use the smallest position value (SPV) rule, and a non-decreasing order mechanism, to convert a position vector of a particle to a job permutation. With the same approach, Tasgetiren et al. (2004a) solve the permutation flow shop problem with makespan and maximum lateness minimization criteria. They hybridize a local search algorithm based on variable neighborhood search (VNS) with a PSO algorithm and show that VNS improves the performance of the PSO algorithm for the proposed research problem.

Pan et al. (2008) propose a discrete PSO algorithm to solve no-wait sequence-dependent flow shop problems with respect to makespan minimization and total completion time minimization. They hybridize discrete PSO with variable neighborhood descent algorithm to improve the solution quality. They also propose several speed-up methods for neighborhood structures. Pan and Wang (2008) present a novel multi-objective PSO algorithm for a no-wait flow shop problem by considering the minimization of makespan and maximum tardiness as bi-objective. Hajinejad et al. (2011) generate a PSO algorithm for group scheduling in a permutation flow shop by considering the minimization of total completion time of jobs.

Lian et al. (2006) proposed a PSO algorithm for permutation flow shop scheduling problems with makespan minimization where the crossover operators are used in PSO. Then, Lian et al. (2008) also proposed a novel PSO algorithm, which used crossover and mutation operators for the same problem. Tasgetiren et al. (2007) combined the PSO algorithm and the Variable Neighborhood Search (VNS) method to minimize the makespan and total flow time in a permutation flow shop sequencing problem. Liu et al. (2008) extended a hybrid PSO algorithm for flow shop scheduling with limited buffers.

Tseng and Liao (2008) perform the research to solve a flexible flow shop scheduling problem by applying the PSO algorithm. They addressed a flexible flow shop scheduling problem with multiprocessor tasks, which means each job has to be processed on several machines in each stage. They develop a regular PSO algorithm to solve the problem with minimization of makespan as the criterion. Tadayon and Salmasi (2013) applied a PSO algorithm for a flexible flow shop problem and show that the algorithm is capable of providing good quality solutions.

Figure 6 demonstrates and compares the types of all scheduling problems, including the types of problems investigated in this research. As stated above, there are many works on group scheduling with respect to the GTAs, but only a few of them focus on group scheduling without the GTAs, i.e., batch scheduling.

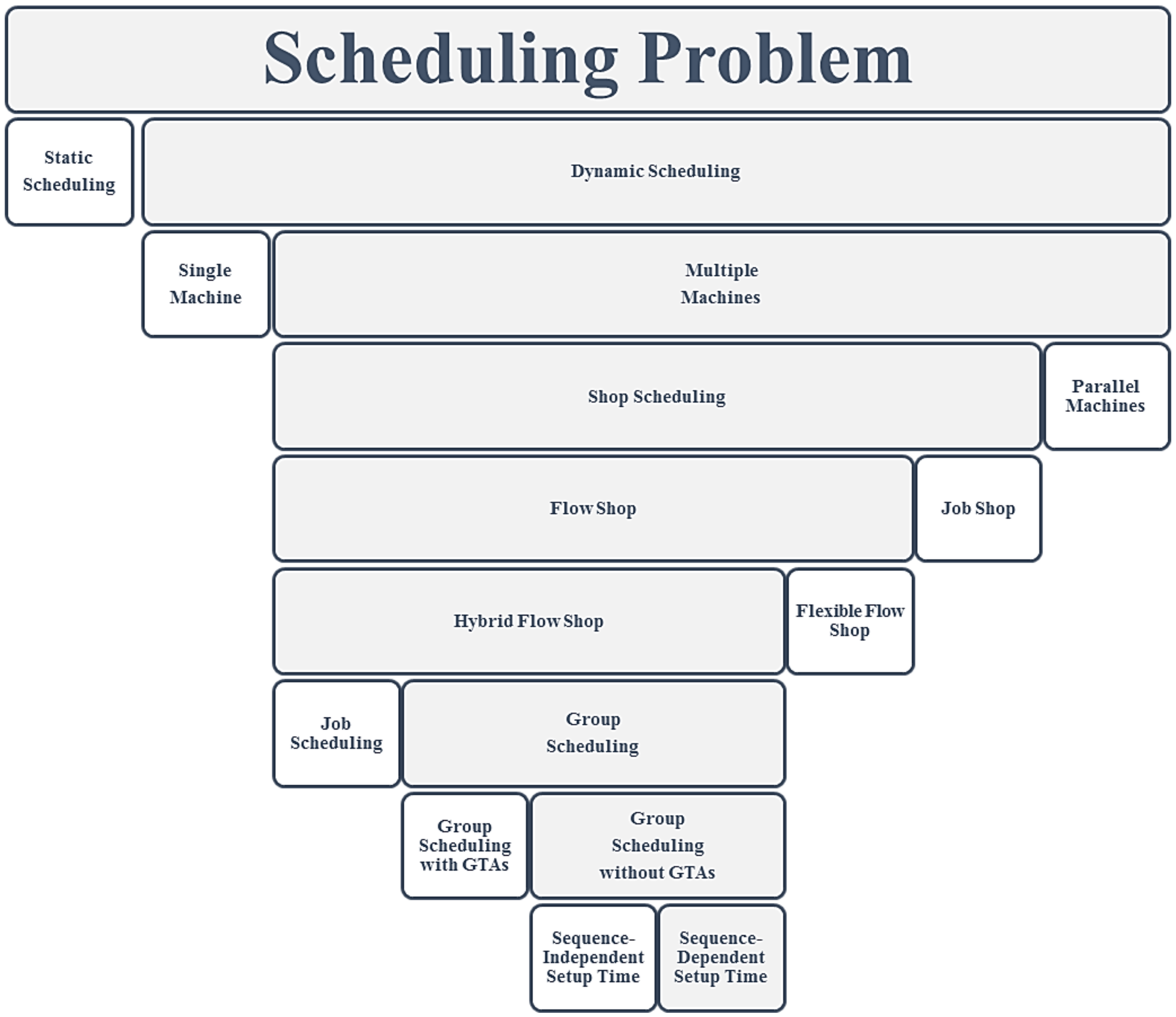


Figure 6. Display of Scheduling Problems

In this research, we are emphasizing the optimization of a HFS with unrelated-parallel machines at least in one stage and sequence-dependent setup times as in most of the previous works. In addition, our research deals with bi-criteria objective function, and dynamic machine availability and job releases times. However, the main characteristic of this research that has never been studied before is splitting jobs belonging to pre-determined groups as inconsistent batches and then scheduling developed batches on machines as well as jobs within each batch to improve the objective function value of group scheduling in HFS. This being the case, a linear mixed-integer programming model, four different types of meta-heuristics, and a method of lower bounding are proposed. To the best of our knowledge, there is no prior work on bi-criteria batching and scheduling problem in HFS that deals with this problem as comprehensively as our research, reported here.

4. METHODOLOGY

The complexity of a problem mostly determines the methodology to deal with the problem. As Gupta and Darrow (1986) mentioned, group scheduling problems in flow shop structures are NP-hard problems, and the problem addressed in this paper is not an exception. There are a number of techniques to prove that a problem is a member of a certain complexity class. Although three techniques including restriction, local replacement, and component design are given by Garey and Johnson (1979) to prove the complexity of a problem, *proof by restriction* is the most applicable and simplest compared to the other two. An NP-hardness proof by restriction for a given problem PϵNP consists of simply showing that P contains a known NP-hard problem Q as a special case and therefore it is NP-hard too. It is worth noting that the main point in this method lies in the specification of the additional restrictions to be placed on the instances of P in such a way that the resulting restricted problem will be identical to Q. Accordingly, as long as there is a one-to-one correspondence between P and Q, it can be concluded that the restricted problem Q is NP-hard and thus the original problem P, which is a harder instance of Q remains NP-hard. The problem addressed in this research is represented by $HF_m\left|ST_{sd,f}, r_j, a_j, M_j, LB_i, skip\right|F_l(\alpha \sum w_j C_j , \beta \sum w_j T_j)$ in the literature of scheduling problems.

Theorem 1. ***Hybrid flow shop batch scheduling problem with minimization of sum of completion time, dynamic job release time, dynamic machine availability time, sequence-dependent setup, machine eligibility, and desired lower bounds is NP-hard in the strong sense.***

Proof: Let P be the hybrid flow shop batch scheduling problem with minimization of sum of completion time, dynamic job release time, dynamic machine availability time, sequence-dependent setup, and machine eligibility ($HF_m\left|ST_{sd,f}, r_j, a_j, M_j, LB_i, skip\right|\sum C_j$). Construct problem Q as a two-machine flow shop scheduling problem with a single machine in each stage and the objective of minimizing the sum of completion time ($F_2||\sum C_j$). Clearly, problem Q is a special case of problem P even when problem P involves more than two machines, since all the setup and run times on the machines other than the first two can be restricted to be 0 in problem P. Problem P can be considered as Q by adjusting the following parameters in the original MILP model:

- the number of batches assigned to each group is restricted to one;
- a desired lower bound is equal to one;
- each batch includes only one job;
- the number of machines is two, so that each stage includes one machine;
- each machine in each stage is identical instead of being unrelated and is capable of processing all jobs;

- all jobs are processed by two machines (no skipping);
- there is no setup time instead of being sequence-dependent setup times;
- the weight of each job is equal to one;
- all jobs and machines are available at the beginning of the planning horizon;

Observe that problem Q is equivalent to the two-machine flow shop scheduling problem with the objective of minimizing the sum of completion time, which has been shown to be strongly NP-hard (Garey et al. 1976). It follows that problem P is NP-hard in the strong sense.

<u>Theorem 2.</u> ***Hybrid flow shop batch scheduling problem with minimization of sum of tardiness, dynamic job release time, dynamic machine availability time, sequence-dependent setup, machine eligibility, and desired lower bounds is NP-hard in the strong sense.***

<u>*Proof*</u>: Let P be the hybrid flow shop batch scheduling problem with minimization of sum of tardiness, dynamic job release time, dynamic machine availability time, sequence-dependent setup, and machine eligibility ($HF_m\left|ST_{sd,f}, r_j, a_j, M_j, LB_i, skip\right|\sum T_j$). Construct problem Q as a single machine scheduling problem with the objective of minimizing the sum of tardiness ($1||\sum T_j$). Clearly, problem Q is a special case of problem P even when problem P involves more than one machine, since all the setup and run times on the machines other than the first one can be restricted to be 0 in problem P. Problem P can be considered as Q by adjusting the following parameters in the original MILP model:

- the number of batches assigned to each group is restricted to one;
- a desired lower bound is equal to one;
- each batch includes only one job;
- the number of machines is restricted to only one single machine (one stage), which is capable of processing all jobs;
- all jobs are processed by the machine (no skipping);
- there is no setup time instead of being sequence-dependent setup times;
- the weight of each job is equal to one;
- all jobs and a machine are available at the beginning of the planning horizon;

Observe that problem Q is equivalent to the single machine scheduling problem with the objective of minimizing the sum of tardiness, which has been shown to be strongly NP-hard (Du and Leung 1990). It follows that problem P is NP-hard in the strong sense.

<u>Theorem 3.</u> ***Hybrid flow shop batch scheduling problem with bi-criteria objective function of minimization of a linear combination of total weighted completion time and total weighted tardiness,***

dynamic job release time, dynamic machine availability time, sequence-dependent setup, machine eligibility, and desired lower bounds is NP-hard in the strong sense.

Proof: Let P be the hybrid flow shop batch scheduling problem with bi-criteria objective function of minimization of a linear combination of total weighted completion time and total weighted tardiness, dynamic job release time, dynamic machine availability time, sequence-dependent setup, and machine eligibility ($HF_m \left| ST_{sd,f}, r_j, a_j, M_j, LB_i, skip \right| F_l(\alpha \sum w_j C_j , \beta \sum w_j T_j)$). Construct problem Q$_1$ as a hybrid flow shop batch scheduling problem with minimization of sum of completion time (HF_m| ... | $\sum C_j$) and construct problem Q$_2$ as a hybrid flow shop batch scheduling problem with minimization of sum of tardiness (HF_m| ... | $\sum T_j$). Both problem Q$_1$ and Q$_2$ follow the same assumptions of problem P. Clearly, problem Q$_1$ and Q$_2$ are special cases of problem P.

Problem P can be considered as Q$_1$ by adjusting the following parameters:

- the weight of individual job's equal to one ($W_j = 1$);
- the sum of weighted tardiness's weight equal to zero ($\beta = 0$).
- Problem P can be considered as Q$_2$ by adjusting the following parameters:
- the weight of individual job's equal to one ($W_j = 1$);
- the sum of weighted completion time's weight equal to zero ($\alpha = 0$).

Problems Q$_1$ and Q$_2$ are proved to be strongly NP-hard based upon Theorems 1 and 2, respectively. It follows that problem P is NP-hard in the strong sense.

Therefore, it can be concluded that the problem investigated in this research is also strongly NP-hard, and there is no guarantee of solving this problem optimally in polynomial time. This remains to be a challenge in solving most complex scheduling problems. In this research two different methodologies are used to deal with the HFS problem, i.e. a mathematical programming model and meta-heuristic algorithms. Since the batch scheduling problem addressed here is among the NP-hard problems, mathematical programming may not be helpful in finding the optimal solution for medium and large size problems. Nevertheless, the mathematical formulation is implemented to represent and communicate with the batch scheduling problem, and evaluate the performance of the heuristic or meta-heuristic algorithms by developing tight lower bounds.

4.1. Mathematical programming model

In this research, four mathematical programming models based on mixed-integer linear programming (MILP) are developed in terms of three approaches used to model scheduling problems.

- The first MILP model, i.e., MILP1, is developed in terms of the precedence constraints between each pair of jobs belonging to the same batch as well as each pair of batches assigned to the same machine (Manne 1960). In this type of formulation, the job sequence is obtained by comparing the sequence of all pairs of jobs within batches. Likewise, the batch sequence is obtained by comparing the sequence of all batches on the same machine. This type of modeling scheduling problems leads to a smaller number of binary variables, but more difficult constraints.
- The second MILP model, i.e., MILP2, considers set of positions for each of the batches and assigns the jobs to those positions. Therefore, the job sequence within batches is determined by finding the job assignment to those positions. Like MILP1, the batch sequence on machines in the MILP2 is determined by the precedence constraints between each pair of batches on the same machines.
- The third MILP model, i.e., MILP3, an adapted version of that proposed by Guinet (1993), is based on the flow conservation constraints to order the jobs on machines.
- Finally, the relaxed MILP model, i.e., RMILP, referred to as the fourth model in this research, is developed as a relaxed version of any of the above MILP models to reduce the solution space of MILP models and find either the optimal solutions or good quality lower bounds within affordable computational time. This relaxed version is obtained by assuming each batch formed from a group is of size 1, and thereby remove the challenge of having to determine the various batch sizes that are applicable to the original MILP model.

Each of these MILP models are described in detail next, based on proposed modeling techniques for scheduling problems.

4.1.1. MILP1

The first MILP model is developed based on the precedence constraints between each pair of jobs and developed batches to mathematically represent the batch scheduling problem and solve small-size problems optimally. This mathematical programming model is developed at two integrated phases including batching and scheduling phases. While the optimal combination of batch compositions is determined by the batching phase, the optimal batch assignment and sequence on machines as well as the optimal job sequence within batches are determined by the scheduling phase, with respect to the batching phase. In the following, first sets, subsets, indices, and parameters that have been used are introduced. Then, all decision variables, including continuous or binary variables, are listed, followed by the mathematical formulation.

Sets and Indices

G	Set of groups, indexed by i, p	$G = \{1,2,\dots,g\}$

G_i	Set of jobs of group i, indexed by j, q	$G_i = \{1,2,\dots,n_i\}$
S_i	Set of batches of group i, indexed by s, t	$S_i = \{1,2,\dots,n_i\}$
K	Set of stages, indexed by k	$K = \{1,2,\dots,m\}$
V^k	Set of machines in stage k, indexed by h	$V^k = \{1,2,\dots,v^k\}$

Subsets

I^k	Subset of groups, which must be processed in stage k	$I^k \subset G$
J_i^k	Subset of jobs of group i, which must be processed in stage k	$J_i^k \subset G_i$
S_i^k	Subset of batches of group i, which can be developed in stage k	$S_i^k \subset S_i$
V_{ij}^k	Subset of machines in stage k, which can process job j of group i	$V_{ij}^k \subset V^k$
K_{ij}	Subset of stages in an ascending order, which must be visited by job j of group i	$K_{ij} \subset K$

Parameters

g	Number of groups
n_i	Number of jobs of group i
n_i^k	Number of jobs of group i, which must be processed in stage k
m	Number of stages
v^k	Number of machines in stage k
m_{ij}	Number of stages, which must be visited by job j of group i
$st_{ij(l)}$	l^{th} stage among subset K_{ij}
t_{ijh}^k	Run time of job j of group i on machine h in stage k
S_{pih}^k	Required setup time to process a batch of group i on machine h in stage k if batch p is the preceding batch ($p = 0$ refers to the reference batch)
d_{ij}	Due date of job j of group i
r_{ij}	Release time of job j of group i
w_{ij}	Weight of job j of group i
a_h^k	Availability time of machine h in stage k
α	Weight attributed to the producer
β	Weight attributed to the customer
LB_{ih}^k	Desired lower bound for the minimum number of jobs assigned to a batch of group i on machine h in stage k

It is worth noting that a desired lower bound, i.e., LB_{ih}^{k}, is determined in terms of a manufacturing company's policies with regard to operating a particular machine to ensure processing a minimum number of jobs. Therefore, LB_{ih}^{k} is not a variable and it is determined before solving the problem.

Decision variables

X_{isj}^{k}	The completion time of job j assigned to batch s of group i in stage k
TD_{ij}	The tardiness of job j of group i
C_{is}^{k}	The completion time of batch s of group i in stage k
$\emptyset_{isj}^{k}$	1 if job j is assigned to batch s of group i in stage k; 0 otherwise
Z_{ish}^{k}	1 if batch s of group i is assigned to machine h in stage k; 0 otherwise
A_{ptis}^{k}	1 if batch s of group i is processed after batch t of group p in stage k; 0 otherwise
Y_{isjq}^{k}	1 if job q is processed after job j assigned to batch s of group i in stage k; 0 otherwise

Mathematical formulation

Based on the precedence constraints to the job sequence within batches and the batch sequence on machines, the MILP1 is developed as follows:

$$Min\, Z = \alpha \sum_{i\in G} \sum_{j\in g_i} \sum_{s\in g_i} w_{ij} X_{isj}^{st_{ij(m_{ij})}} + \beta \sum_{i\in G} \sum_{j\in g_i} w_{ij} TD_{ij} \tag{4.1}$$

The objective function (4.1) minimizes a linear combination of total weighted completion time and total weighted tardiness. α and β are the weights attributed to producer and customers, respectively, and are normalized with the help of the following equation: $\alpha + \beta = 1$. Set of constraints (4.2) through (4.5), known as batching constraint sets, are incorporated into the model to determine the optimal batch composition of each group in each stage.

$$\sum_{s\in S_i^k} \emptyset_{isj}^{k} = 1 \qquad i \in I^k;\ j \in J_i^k;\ k \in K; \tag{4.2}$$

$$\sum_{h\in V^k} Z_{ish}^{k} \le 1 \qquad i \in I^k;\ s \in S_i^k;\ k \in K; \tag{4.3}$$

$$\sum\nolimits_{h \in V^k} Z_{ish}^k \geq \emptyset_{isj}^k \tag{4.4}$$
$$i \in I^k;\ j \in J_i^k;\ s \in S_i^k;\ k \in K;$$

$$\sum\nolimits_{j \in J_i^k} \emptyset_{isj}^k \geq \sum\nolimits_{h \in V^k} \left(LB_{ih}^k\right) Z_{ish}^k \tag{4.5}$$
$$s \in S_i^k;\ i \in I^k;\ k \in K;$$

A batch composition of group i with J_i^k jobs in k^{th} stage represents the number of batches assigned to group i as well as the number and type of jobs belonging to each batch of group i, with respect to the desired lower bounds on batch sizes (LB_{ih}^k). In other words, n_i^k jobs should be assigned to s batches processed on one or more machines in k^{th} stage, where $s \in \{1, \max_{h \in v^k} \lceil n_i^k / LB_{ih}^k \rceil\}$. Apart from this, in each stage of a hybrid flow shop, not only each job of a group (which is not skipping the stage) must be assigned to one and only one batch of its group (constraint (4.2)), but also a batch including at least one job must be assigned to one and only one machine (constraints (4.3) and (4.4)). Constraint (4.5) ensures that the minimum number of jobs assigned to a batch should be equal or greater than its lower bound on a related machine.

Set of constraints (4.6) through (4.13), known as scheduling constraint sets, are incorporated into the model to determine the optimal batch sequence on machines and job sequence within batches, with regard to the batching phase.

$$X_{isj}^k + M\left(1 - A_{ptis}^k\right) + M\left(1 - Z_{ish}^k\right) + M\left(1 - Z_{pth}^k\right) + M\left(1 - \emptyset_{isj}^k\right) \geq C_{pt}^k + S_{pih}^k + t_{ijh}^k \tag{4.6}$$
$$i, p \in I^k (p \leq i; p = i \rightarrow t < s);\ j \in J_i^k;\ h \in v_{ij}^k\ ;\ k \in K;\ s \in S_i^k;\ t \in g_p^k;\ M{:}\, large\ number;$$

$$X_{ptj}^k + M\left(A_{ptis}^k\right) + M\left(1 - Z_{ish}^k\right) + M\left(1 - Z_{pth}^k\right) + M\left(1 - \emptyset_{ptj}^k\right) \geq C_{is}^k + S_{iph}^k + t_{pjh}^k \tag{4.7}$$
$$i, p \in I^k (p \leq i; p = i \rightarrow t < s);\ j \in J_p^k;\ h \in v_{pt}^k;\ k \in K;\ s \in S_i^k;\ t \in g_p^k;\ M{:}\, large\ number;$$

The sequence of batches is determined in each stage by constraints (4.6) and (4.7), so that they *simultaneously* assign values to binary variables A_{ptis}^k. These constraints restrict the completion time of each job belonging to each batch of each group (which is not skipping the stage) to be greater than the completion time of the previous batch plus the sequence-dependent machine setup time, and the required run time for processing the job on a particular stage.

$$X_{isj}^{k} + M\left(1 - \emptyset_{isj}^{k}\right) \geq \sum\nolimits_{h \in v_{ij}^{k}} \left(a_{h}^{k} + S_{0ih}^{k} + t_{ijh}^{k}\right) Z_{ish}^{k} \tag{4.8}$$

$$i \in I^{k};\ j \in J_{i}^{k}; s \in S_{i}^{k};\ k \in K;$$

$$X_{isj}^{st_{ij(1)}} + M\left(1 - \emptyset_{isj}^{st_{ij(1)}}\right) \geq r_{ij} + \sum_{h \in v_{ij}^{st_{ij(1)}}} t_{ijh}^{st_{ij(1)}} \left(Z_{ish}^{st_{ij(1)}}\right) \tag{4.9}$$

$$i = 1,2,\dots,g; j = 1,2,\dots,n_{i};\ s \in S_{i}^{st_{ij(1)}};$$

Constraints (4.8) together with constraint (4.9) account for dynamic machine availability and dynamic job release time, respectively. In other words, constraint (4.8) ensures that a job cannot to be processed if the assigned machine is not available in each stage, while constraint (4.9) ensures that the processing of each job cannot be started if the job is not released.

$$X_{isj}^{k} - X_{isq}^{k} + M\left(Y_{isjq}^{k}\right) + M\left(1 - \emptyset_{isj}^{k}\right) + M\left(1 - \emptyset_{isq}^{k}\right) \geq \sum\nolimits_{h \in v_{ij}^{k} \cap v_{iq}^{k}} t_{ijh}^{k}\left(Z_{ish}^{k}\right) \tag{4.10}$$

$$i \in I^{k};\ j,q \in J_{i}^{k}(j < q); k \in K; s \in S_{i}^{k};\ M\text{: }large\ number;$$

$$X_{isq}^{k} - X_{isj}^{k} + M\left(1 - Y_{isjq}^{k}\right) + M\left(1 - \emptyset_{isj}^{k}\right) + M(1 - \emptyset_{isq}^{k}) \geq \sum\nolimits_{h \in v_{ij}^{k} \cap v_{iq}^{k}} t_{iqh}^{k}(Z_{ish}^{k}) \tag{4.11}$$

$$i \in I^{k};\ j,q \in J_{i}^{k}(j < q); k \in K; s \in S_{i}^{k};\ M\text{: }large\ number;$$

The sequence of jobs within batches are determined in each stage by constraints (4.10) and (4.11), so that they *simultaneously* assign values to binary variables Y_{isjq}^{k}. These constraints ensure that, in each stage, the difference between the completion times of any two jobs within a batch (which are not skipping the stage) is equal or greater than the run time of the succeeding job.

$$C_{is}^{k} \geq X_{isj}^{k} \tag{4.12}$$

$$i \in I^{k};\ j \in J_{i}^{k};\ s \in S_{i}^{k}; k \in K$$

The completion time of each batch of each group in each stage is determined by constraint (4.12), so that it should be greater than the completion times of all of its jobs, if it is not skipping that stage.

$$X_{isj}^{st_{ij(l)}} - X_{is'j}^{st_{ij(l-1)}} \geq \sum_{h \in v_{ij}^{st_{ij(r)}}} t_{ijh}^{st_{ij(l)}} \left(Z_{ish}^{st_{ij(l)}}\right) + \left(\phi_{isj}^{st_{ij(l)}} + \phi_{is'j}^{st_{ij(l-1)}} - 2\right) M \tag{4.13}$$

$$i = 1,2,\dots,g; j = 1,2,\dots,n_i;\ s \in S_i^{st_{ij(r)}}; s' \in S_i^{st_{ij(r-1)}}; l \in \{2,3,\dots,m_{ij}\};$$

Constraint (4.13) in scheduling constraint sets ensures that the operation of each job in each stage cannot be started until it has been completely processed on a prior stage. This prior stage is where the job had its latest operation.

$$TD_{ij} \geq X_{isj}^{st_{ij(m_{ij})}} - d_{ij} \tag{4.14}$$

$$i = 1,2,\dots,g; j = 1,2,\dots,n_i;\ s \in S_i^{st_{ij(m_{ij})}};$$

Constraint (4.14) is applied to find the tardiness of each job, which should be greater than or equal to both the completion time minus due date and zero.

$$X_{isj}^{k}, TD_{ij}, C_{is}^{k} \geq 0;$$

$$Z_{ish}^{k} \in \{0,1\}; A_{ptis}^{k} \in \{0,1\}\ (p \leq i; p = i \rightarrow t < s); Y_{isjq}^{k} \in \{0,1\}\ (j < q); \emptyset_{isj}^{k} \in \{0,1\}; \tag{4.15}$$

$$i \in G; j \in G_i; k \in K; i,p \in I^k;\ j,q \in J_i^k;\ s \in S_i^k;\ t \in g_p^k; h \in v_{ij}^k;\ M\text{: large number.}$$

Constraint (4.15) defines the variables used.

4.1.2. MILP2

The second MILP model is very similar to the MILP1, except that it follows the position concept within batches to determine the job sequence within batches in the scheduling phase. The main reason for developing this model is to eliminate the precedence constraints related to the job sequence within batches. If this were the case, the MILP1 is improved by reducing the number of constraints and variables and, consequently, the model complexity. The sets, subsets, indices, parameters, decision variables, and mathematical formulation for the MILP2 are presented next.

Sets and Indices

G	Set of groups, indexed by i, p	$G = \{1,2,\dots,g\}$

G_i	Set of jobs of group i, indexed by j, q	$G_i = \{1,2, \dots, n_i\}$
S_i	Set of batches of group i, indexed by s, t	$S_i = \{1,2, \dots, n_i\}$
K	Set of stages, indexed by k	$K = \{1,2, \dots, m\}$
V^k	Set of machines in stage k, indexed by h	$V^k = \{1,2, \dots, v^k\}$

Subsets

I^k	Subset of groups, which must be processed in stage k	$I^k \subset G$
J_i^k	Subset of jobs of group i, which must be processed in stage k	$J_i^k \subset G_i$
S_i^k	Subset of batches of group i, which can be developed in stage k	$S_i^k \subset S_i$
V_{ij}^k	Subset of machines in stage k, which can process job j of group i	$V_{ij}^k \subset V^k$
K_{ij}	Subset of stages in an ascending order, which must be visited by job j of group i	$K_{ij} \subset K$

Parameters

g	Number of groups
n_i	Number of jobs of group i
n_i^k	Number of jobs of group i, which must be processed in stage k
m	Number of stages
v^k	Number of machines in stage k
m_{ij}	Number of stages, which must be visited by job j of group i
$st_{ij(l)}$	l^{th} stage among subset K_{ij}
t_{ijh}^k	Run time of job j of group i on machine h in stage k
S_{pih}^k	Required setup time to process a batch of group i on machine h in stage k if a batch of group p is the preceding batch ($p = 0$ refers to the reference batch)
d_{ij}	Due date of job j of group i
r_{ij}	Release time of job j of group i
w_{ij}	Weight of job j of group i
a_h^k	Availability time of machine h in stage k
α	Weight attributed to the producer
β	Weight attributed to the customer
LB_{ih}^k	Desired lower bound for the minimum number of jobs assigned to a batch of group i on machine h in stage k

Decision variables

X_{isj}^{k}	The completion time of job j assigned to batch s of group i in stage k
TD_{ij}	The tardiness of job j of group i
C_{is}^{k}	The completion time of batch s of group i in stage k
$\emptyset_{isrj}^{k}$	1 if job j assigned to r^{th} position of batch s of group i in stage k; 0 otherwise
Z_{ish}^{k}	1 if batch s of group i is assigned to machine h in stage k; 0 otherwise
A_{ptis}^{k}	1 if batch s of group i is processed after batch t of group p in stage k; 0 otherwise

Mathematical formulation

Based on the precedence constraints for the batch sequence on machines, the MILP2 is developed as follows:

$$Min\ Z = \alpha \sum_{i\in G} \sum_{j\in G_i} \sum_{s\in S_i} w_{ij} X_{isj}^{st_{ij(m_{ij})}} + \beta \sum_{i\in G} \sum_{j\in G_i} w_{ij} TD_{ij} \qquad (4.16)$$

A linear combination of total weighted completion time and total weighted tardiness of jobs is considered as the objective function presented in (4.16). Set of constraints (4.17) through (4.22), known as batching constraint sets, are incorporated into the model to determine the optimal combination of batch compositions. Regarding LB_{ih}^{k}, the range of possible developed batches of group i in stage k is $\{1, \max_{h\in V^k} \lfloor n_i^k / LB_{ih}^k \rfloor\}$.

$$\sum_{s\in S_i^k} \sum_{r\in J_i^k} \emptyset_{isrj}^{k} = 1$$
$$i \in I^k; j \in J_i^k; k \in K; \qquad (4.17)$$

$$\sum_{j\in J_i^k} \emptyset_{isrj}^{k} \geq \sum_{j\in J_i^k} \emptyset_{is(r+1)j}^{k}$$
$$i \in I^k; s \in S_i^k; r \in J_i^k - \{1\}; k \in K; \qquad (4.18)$$

It is assumed that n_i^k positions exist for each batch of group i in stage k. Therefore, in each stage of HFS, not only each job of a group (which is not skipping the stage) must be assigned to one and only one position available for only one batch of its group (constraint (4.17)), but also the jobs that belong to a batch must be assigned to the consecutive positions available for that batch from the first position (constraint (4.18)).

$$\sum\nolimits_{j\in J_i^k} \emptyset_{isrj}^k \le 1 \qquad i\in I^k; s\in S_i^k; r\in J_i^k; k\in K; \tag{4.19}$$

$$\sum\nolimits_{h\in V^k} Z_{ish}^k \le 1 \qquad i\in I^k; s\in S_i^k; k\in K; \tag{4.20}$$

$$\sum\nolimits_{h\in V^k} Z_{ish}^k \ge \sum\nolimits_{r\in J_i^k} \emptyset_{isrj}^k \qquad i\in I^k; j\in J_i^k; s\in S_i^k; k\in K; \tag{4.21}$$

$$\sum\nolimits_{j\in J_i^k}\sum\nolimits_{r\in J_i^k} \emptyset_{isrj}^k \ge \sum\nolimits_{h\in V^k} \left(LB_{ih}^k\right) Z_{ish}^k \qquad i\in I^k; s\in S_i^k; k\in K; \tag{4.22}$$

The other constraints of batching constraint sets (constraints (4.19) through (4.22)) are incorporated into the model to assign a batch including at least one job to a machine, with respect to LB_{ih}^k. Set of constraints (4.23) through (4.29), known as scheduling constraint sets, are incorporated into the model to determine the optimal batch assignment and sequence on machines as well as job sequence within batches, with regard to the batching phase.

$$\begin{aligned} X_{isj}^k + M\left(1-A_{ptis}^k\right) + M\left(1-Z_{ish}^k\right) + M\left(1-Z_{pth}^k\right) + M\left(1-\sum\nolimits_{r\in J_i^k} \emptyset_{isrj}^k\right) \\ \ge C_{pt}^k + S_{pih}^k + t_{ijh}^k \end{aligned} \tag{4.23}$$

$$i,p\in I^k (p\le i; if\ p=i\to t\ne s); j\in J_i^k; h\in V_{ij}^k; k\in K; s\in S_i^k; t\in S_p^k;$$

$$X_{ptj}^k + M\left(A_{ptis}^k\right) + M\left(1-Z_{ish}^k\right) + M\left(1-Z_{pth}^k\right) + M\left(1-\sum\nolimits_{r\in J_i^k} \emptyset_{ptrj}^k\right) \ge C_{is}^k + S_{iph}^k + t_{pjh}^k \tag{4.24}$$

$$i,p \in I^k (p \leq i; if\ p = i \rightarrow t \neq s); j \in J_p^k; h \in V_{pj}^k; k \in K; s \in S_i^k; t \in S_p^k;$$

Constraint (4.23) together with constraint (4.24) are incorporated to find the sequence of batches on machines (A_{ptis}^k).

$$X_{isj}^k - X_{isq}^k + M\left(1 - \sum\nolimits_{r \in J_i^k} \emptyset_{isrj}^k\right) + M\left(1 - \sum\nolimits_{r \in J_i^k} \emptyset_{is(r-1)q}^k\right) \geq \sum\nolimits_{h\ \in V_{ij}^k \cap V_{iq}^k} \left(t_{ijh}^k \times Z_{ish}^k\right) \tag{4.25}$$
$$i \in I^k; j,q \in J_i^k (j < q); s \in S_i^k; r \in J_i^k - \{1\}; k \in K;$$

Constraint (4.25) is incorporated to find the sequence of jobs within each batch ($\emptyset_{isrj}^k$).

$$X_{isj}^k + M\left(1 - Z_{ish}^k\right) + M\left(1 - \sum\nolimits_{r \in J_i^k} \emptyset_{isrj}^k\right) \geq a_h^k + S_{0ih}^k + \sum\nolimits_{r \in J_i^k} \left(\emptyset_{isrj}^k \times t_{ijh}^k\right) \tag{4.26}$$
$$i \in I^k; j \in J_i^k; s \in S_i^k; h \in V_{ij}^k; k \in K;$$

$$X_{isj}^{st_{ij(1)}} + M\left(1 - Z_{ish}^{st_{ij(1)}}\right) + M\left(1 - \sum\nolimits_{r \in J_i^{st_{ij(1)}}} \emptyset_{isrj}^{st_{ij(1)}}\right) \geq r_{ij} + t_{ijh}^{st_{ij(1)}} \tag{4.27}$$
$$\forall i \in G; j \in G_i; s \in S_i^{st_{ij(1)}}; h \in V_{ij}^{st_{ij(1)}};$$

Constraint (4.26) together with constraint (4.27) account for dynamic machine availability and dynamic job release time, respectively.

$$C_{is}^k \geq X_{isj}^k \tag{4.28}$$
$$i \in I^k;\ j \in J_i^k;\ s \in S_i^k; k \in K;$$

$$X_{isj}^{st_{ij(l)}} - X_{is'j}^{st_{ij(l-1)}} + M\left(1 - Z_{ish}^{st_{ij(l)}}\right) \geq t_{ijh}^{st_{ij(l)}} + \left(\sum\nolimits_{r=1}^{J_i^{st_{ij(l)}}} \emptyset_{isrj}^{st_{ij(l)}} + \sum\nolimits_{r=1}^{J_i^{st_{ij(l-1)}}} \emptyset_{is'rj}^{st_{ij(l-1)}} - 2\right) M \tag{4.29}$$
$$i \in G; j \in G_i; s \in S_i^{st_{ij(l)}}; s' \in S_i^{st_{ij(l-1)}}; h \in V_{ij}^{st_{ij(l)}}; l \in \{2,3,\dots,m_{ij}\};$$

Constraint (4.28) determines the completion time of each batch in each stage, while constraint (4.29), known as the linking constraint, ensures that there is a connection between completion times of a job related to each of two sequential stages, where the job had operations.

$$TD_{ij} \geq X_{isj}^{st_{ij(m_{ij})}} - d_{ij}$$
$$i \in G; j \in G_i; s \in S_i^{st_{ij(m_{ij})}}; \tag{4.30}$$

Constraint (4.30) is applied for finding the tardiness of each job.

$$X_{isj}^{k}, TD_{ij}, C_{is}^{k} \geq 0;$$
$$Z_{ish}^{k} \in \{0,1\}; A_{ptis}^{k} \in \{0,1\}(p \leq i \; and \; if \; p = i \; then \; t < s); \emptyset_{isrj}^{k} \in \{0,1\} \tag{4.31}$$
$$i \in G; j \in G_i; i,p \in I^k; j,q \in J_i^k; s \in S_i^k; t \in S_p^k; r \in J_i^k; h \in V_{ij}^k; k \in K; M\text{: large number.}$$

Finally, constraint (4.31) defines the variables used.

4.1.3. MILP3

The third MILP model, an adapted version of that proposed by Guinet (1993), is developed in terms of the flow conservation constraints to order the jobs within the machines, while respecting the desired lower bounds on batch sizes. Therefore, the MILP1 and MILP2 models are improved by integrating variables, which indicate job assignment to batches, batch assignment on machines, job sequence within batches, and batch sequence on machines. Thus, batching and scheduling phases are integrated in the MILP3 model. The following sets and indices for the MILP3 are defined.

Sets and Indices

G	Set of groups, indexed by i	$G = \{1,2,\ldots,g\}$
N	Set of jobs (of all groups), indexed by l, j	$N = \{1,2,\ldots,n\}$
K	Set of stages, indexed by k	$K = \{1,2,\ldots,m\}$
V^k	Set of machines in stage k, indexed by h	$V^k = \{1,2,\ldots,v^k\}$

The group and batch indices of the MILP1 and MILP2 models are eliminated from the MILP3 model to reduce the complexity of the model. And, instead, the set of indices of job numbers is determined in terms of the ascending order of group numbers. Then, the results obtained by the MILP3 model can be interpreted as batch scheduling by considering the consecutive jobs of the same group in the schedule as batches. The set of indices of job numbers related to i^{th} group is determined as follows:

Sets and Indices (Cont.)

G_i	Set of jobs of group i, indexed by l, j	$G_i = \left[1 + \sum_{g=1}^{i-1} n_g, 2 + \sum_{g=1}^{i-1} n_g, \dots, \sum_{g=1}^{i} n_g\right]$

where n_i $(i \in G)$ indicate the number of jobs of group i. Thus, the group index is removed from the following subsets and parameters.

Subsets

I^k	Subset of groups, which must be processed in stage k	$I^k \subset G$
N^k	Subset of jobs, which must be processed in stage k	$N^k \subset N$
V_j^k	Subset of machines in stage k, which can process job j	$V_j^k \subset V^k$
K_j	Subset of stages in an ascending order, which must be visited by job j	$K_j \subset K$

Parameters

g	Number of groups
n_i	Number of jobs of group i
n	Number of jobs of all groups, $\sum_{i \in G} n_i = n$
m	Number of stages
v^k	Number of machines in stage k
m_j	Number of stages, which must be visited by job j
$st_{j(r)}$	r^{th} stage among subset K_j
t_{jh}^k	Run time of job j on machine h in stage k
S_{ljh}^k	Required setup time to process job j on machine h in stage k if job l is the preceding job ($l = 0$ refers to the reference job)
d_j	Due date of job j
r_j	Release time of job j
w_j	Weight of job j
a_h^k	Availability time of machine h in stage k
α	Weight attributed to the producer
β	Weight attributed to the customer

LB_{ih}^{k}	Desired lower bound for the minimum number of jobs assigned to a batch of group i on machine h in stage k

A machine- and sequence-dependent setup time S_{ljh}^{k} $(l, j \in N^k | l \neq j, h \in V_l^k \cap V_j^k$, and $k \in M)$ is incurred whenever $l, j \notin G_i$ $(i \in G)$; $S_{ljh}^{k} = 0$, otherwise.

Decision variables

x_{ljh}^{k}	1 if job j is scheduled immediately after job l on machine h in stage k; 0 otherwise
$Y_{j_1 j_2 \cdots j_l}^{ihk}$	1 if for a LB_{ih}^{k} jobs of group i on machine h in stage k, there is at least one sequence including at least two jobs; 0 otherwise
X_j^k	The completion time of job j in stage k
T_j	The tardiness of job j

The variable $x_{0jh}^{k}/x_{j(n+1)h}^{k} = 1$ if the first/last job processed by machine h is job j in stage k; $x_{0jh}^{k}/x_{j(n+1)h}^{k} = 0$, otherwise.

Mathematical formulation

Based on the flow conservation constraints to the job sequence within machines, the MILP3 model is developed as follows:

$$Min\ Z = \alpha \sum_{j \in N} w_j X_j^{st_{j(m_j)}} + \beta \sum_{j \in N} w_j T_j \tag{4.32}$$

The objective function (4.32) is to *simultaneously* minimize the total weighted completion time and total weighted tardiness.

$$\sum_{\substack{l \in N^k \cup \{0\} \\ l \neq j}} \sum_{h \in V_l^k \cap V_j^k} x_{ljh}^{k} = 1 \qquad \forall j \in N^k, \forall k \in M, \tag{4.33}$$

Constraint (4.33) ensures that, in each stage of a hybrid flow shop, each job (which is not skipping the stage) is assigned to exactly one machine with exactly one predecessor. The dummy jobs 0 are applied for the first job processed on a machine, i.e., x_{0jh}^{k}.

$$\sum_{j \in N^k} x_{0jh}^{k} \leq 1 \qquad \forall h \in V^k, \forall k \in M, \tag{4.34}$$

Constraint (4.34) guarantees that each machine is used at most once.

$$\sum_{\substack{j \in N^k \\ l \neq j}} \sum_{h \in V_l^k \cap V_j^k} x_{ljh}^{k} \leq 1 \qquad \forall l \in N^k, \forall k \in M, \tag{4.35}$$

Constraint (4.35) limits the maximum number of successors of every job (which is not skipping the stage) to one, on each machine in each stage. The jobs must be properly linked in each machine so that if a given job l is processed on a given machine h, a predecessor u must exist on the same machine.

$$\sum_{\substack{l \in N \cup \{0\} \\ l \neq j}} x_{ljh}^{k} = \sum_{\substack{l \in N \cup \{n+1\} \\ l \neq j}} x_{jlh}^{k} \qquad \forall j \in N^k, \forall h \in V_j^k, \forall k \in M \tag{4.36}$$

Constraint (4.36), known as flow conservation constraint set, is incorporated into the model to ensure that the jobs are properly linked within a machine schedule.

Assume Q_{ih}^{k} $(\forall k \in M, \forall\ h \in V^k, \forall i \in I^k | LB_{ih}^{k} > 1)$ is a set of all possible combinations of ℓ different jobs j_1, j_2, …, and j_l belonging to group i, which can be processed by machine h in stage k, where $\ell = LB_{ih}^{k}$. The number of members of Q_{ih}^{k} is ${}_{n_{ih}^{k}}C_{\ell} = \binom{n_{ih}^{k}}{\ell}$, where n_{ih}^{k} represents the number of jobs in group i, which can be processed by machine h in stage k. Then, define the binary variables $Y_{j_1 j_2 \cdots j_l}^{ihk}$ for each member of Q_{ih}^{k} $(\forall k \in M, \forall\ h \in V^k, \forall i \in I^k)$, which is composed of ℓ different jobs j_1, j_2, …, and j_l. For each member of Q_{ih}^{k}, i.e., $\{j_1, j_2, \dots, j_l\} \in Q_{ih}^{k} | j_1 \neq j_2 \neq \cdots \neq j_\ell$, the value of the binary variable $Y_{j_1 j_2 \dots j_l}^{ihk}$ is automatically set in the model as follows:

$$Y_{j_1 j_2 \dots j_l}^{ihk} \begin{cases} 1; \text{ if } \exists j, j' \in \{j_1, j_2, \dots, j_l\} | x_{jj'h}^{k} = 1 \text{ or } x_{j'jh}^{k} = 1 \\ 0; \text{ otherwise} \end{cases}$$

For each member of Q_{ih}^{k}, the number of subscript indices of $Y_{j_1 j_2 \dots j_l}^{ihk}$ is equal to ℓ. For example, for the first group with 4 jobs and $LB_{1h}^{k} = 2$, the set of binary variables $\{Y_{12}^{1hk}, Y_{13}^{1hk}, Y_{14}^{1hk}, Y_{23}^{1hk}, Y_{24}^{1hk}, Y_{34}^{1hk}\}$ corresponds to the members of $Q_{1h}^{k} = \{(1,2), (1,3), (1,4), (2,3), (2,4), (3,4)\}$, on machine h.

$$\left(\sum_{j=j_1}^{j_\ell} \sum_{l \in g_i | l \neq j_1, l \neq j_2, \dots, l \neq j_\ell} \left(x_{ljh}^{k} + x_{jlh}^{k} \right) \right) + 2 \left(\sum_{j=j_1}^{j_\ell} \sum_{j'=j+1}^{j_\ell} \left(x_{jj'h}^{k} + x_{j'jh}^{k} \right) \right) \geq 2\left(Y_{j_1 j_2 \dots j_l}^{ihk}\right)\left(LB_{ih}^{k} - 1\right) \tag{4.37}$$

$$M\left(Y_{j_1 j_2 \dots j_l}^{ihk}\right) \geq \sum_{j=j_1}^{j_\ell} \sum_{j'=j+1}^{j_\ell} \left(x_{jj'h}^{k} + x_{j'jh}^{k}\right) \tag{4.38}$$

$$M\left(Y_{j_1 j_2 \dots j_l}^{ihk} - 1\right) \leq \sum_{j=j_1}^{j_\ell} \sum_{j'=j+1}^{j_\ell} \left(x_{jj'h}^{k} + x_{j'jh}^{k}\right) - \varepsilon\left(Y_{j_1 j_2 \dots j_l}^{ihk}\right) \tag{4.39}$$

$$\forall k \in M, \forall\, h \in V^{k}, \forall i \in I^{k} | LB_{ih}^{k} = \ell \ (\ell > 1), \forall \{j_1, j_2, \dots, j_l\} \in Q_{ih}^{k}, 0 < \varepsilon < 1$$

Constraints (4.37) through (4.39), known as desired lower bounds constraint sets, are incorporated into the model to establish a balance between processing and setup times of each batch formed on a machine in each stage. These constraints work jointly for each combination of parameters. Basically, if the desired lower bounds on batch sizes for group i ($i \in I^{k}$) on machine h ($h \in V^{k}$) in stage k ($k \in M$) is equal to ℓ (i.e., $LB_{ih}^{k} = \ell$), these constraints limit the minimum number of jobs assigned to each batch of group i, which is formed on machine h in stage k to ℓ. This being the case, this set of constraints indicate that the number of sequential jobs of a group assigned to a machine must be equal or greater than LB_b of the group on that machine. The functionality of constraints (4.37) through (4.39) is further explained in the next sub-section using an example problem.

$$X_j^k + M\left(1 - x_{ljh}^k\right) \geq X_l^k + S_{ljh}^k + t_{jh}^k$$
$$\forall l \in N^k \cup \{0\}, \forall j \in N^k | l \neq j, \forall h \in V_l^k \cap V_j^k, \forall k \in M, \tag{4.40}$$

Constraint (4.40) is incorporated to control the completion times of the jobs at the machines in each stage. This constraint restricts the completion time of job j (C_j^k) to be greater than the completion time of job l (C_l^k), plus the setup time between jobs l and j (S_{ljh}^k), and the run time of job l (t_{lh}^k), if job j is processed immediately after job l on machine h in k^{th} stage (i.e., $x_{ljh}^k = 1$). If $x_{ljh}^k = 0$, then the big constant M renders the constraint redundant.

$$X_0^k = a_h^k$$
$$\forall h \in V^k, \forall k \in M, \tag{4.41}$$

$$X_j^{st_{j(1)}} \geq r_j + \sum_{\substack{l \in N^k \cup \{0\} \\ j \neq l}} \sum_{h \in V_l^k \cap V_j^k} t_{jh}^{st_{j(1)}} \left(x_{ljh}^{st_{j(1)}}\right)$$
$$\forall\, j \in N^k, \tag{4.42}$$

Constraints (4.41) and (4.42) account for dynamic machine availability time and dynamic job release time, respectively. They ensure that a job can be processed by a machine only when the job is released to the machine and the assigned machine is available.

$$X_j^{st_{j(r)}} - X_j^{st_{j(r-1)}} \geq \sum_{\substack{l \in N^k \cup \{0\} \\ j \neq l}} \sum_{h \in V_l^k \cap V_j^k} t_{jh}^{st_{j(r)}} \left(x_{ljh}^{st_{j(r)}}\right)$$
$$\forall j \in N^k, r \in \{2,3, \dots, m_j\} \tag{4.43}$$

Constraint (4.43) ensures that the operation of each job in each stage cannot be started until it has been completely processed on a prior stage, where the job had its latest operation.

$$T_j \geq X_j^{st_{j(m_j)}} - d_j \tag{4.44}$$

$$\forall j \in N,$$

Constraint (4.44) determines the tardiness of a job, which is equal or greater than both the completion time on the last stage minus due date, and zero.

$$X_j^k \geq 0$$

$$\forall j \in N^k, \forall k \in M,$$

$$T_j \geq 0 \qquad (4.45)$$

$$\forall j \in N,$$

$$x_{ljh}^k \in \{0, 1\}$$

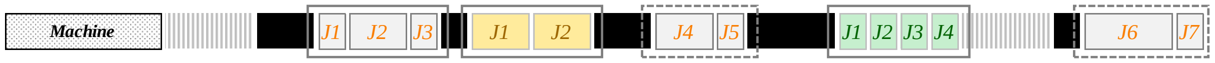
$$\forall l \in N^k \cup \{0\}, \forall j \in N^k \cup \{n+1\} | l \neq j, \forall h \in V_l^k \cap V_j^k, \forall k \in M.$$

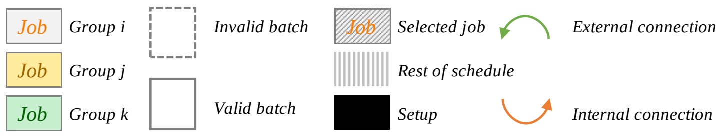
Finally, constraint (4.45) defines the real and integer requirements imposed on the variables.

4.1.3.1. Functionality of desired lower bounds constraints

The desired lower bounds on batch sizes must be satisfied by the jobs assigned and arranged on machines. This being the case, each member of Q_{ih}^k $(\forall k \in M, \forall h \in V^k, i \in I^k | LB_{ih}^k > 1)$ including at least one internal connection (IC) in the sequence must satisfy constraint (4.37); otherwise there is a violation on LB_{ih}^k. If at least a pair of jobs j and j' belonging to $\{j_1, j_2, \dots, j_l\}$ satisfies $x_{jj'}^h = 1$ or $x_{j'j}^h = 1$, there is at least one IC between jobs belonging to $\{j_1, j_2, \dots, j_l\}$. The maximum number of ICs between jobs belonging to a selected member of Q_{ih}^k is $(LB_{ih}^k - 1)$. If there is not any IC between jobs belonging to $\{j_1, j_2, \dots, j_l\}$, there are at least three and at most six external connections (ECs) related to those jobs in the sequence. An EC is a connection between job $j \in \{j_1, j_2, \dots, j_l\}$ and job $l \in G_i | l \notin \{j_1, j_2, \dots, j_l\}$, i.e., $x_{jl}^h = 1$ or $x_{lj}^h = 1$.

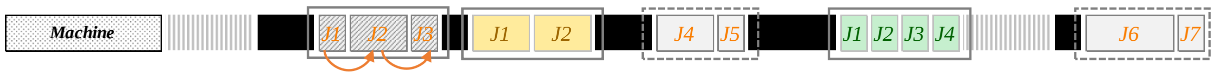
In order to show functionality of constraints (4.37) through (4.39), consider the following job sequence of group i with 7 jobs and $LB_{ih}^k = 3$ on a particular machine h in stage k and the Gantt chart below as a guide.





The batch including jobs 4 and 5 as well as the batch including jobs 6 and 7 violate LB_{ih}^{k}, while the batch including jobs 1, 2, and 3 satisfies LB_{ih}^{k}. The first and second parts of the left-hand side in equation (4.37) enumerate the number of ECs and ICs related to a selected member of Q_{ih}^{k}, respectively. Equations (4.38) and (4.39) are accompanied by equation (4.37) so that if there is any IC between a selected member of Q_{ih}^{k}, equation (4.37) must be satisfied by that member; otherwise equation (4.37) is considered as a redundant constraint. This being the case, equations (4.38) and (4.39) determine the binary variable corresponding to $\{j_1, j_2, \dots, j_l\} \in Q_{ih}^{k}$ as $Y_{j_1 j_2 \dots j_l}^{ihk} = 1$ if there is at least one IC between jobs belonging to $\{j_1, j_2, \dots, j_l\}$; otherwise $Y_{j_1 j_2 \dots j_l}^{ihk} = 0$. In the following, with the help of three scenarios, we explain the relationship between these constraints.

Scenario 1: consider the following selected jobs, i.e., $\{1, 2, 3\} \in Q_{ih}^{k}$:



$$\sum_{j=j_1}^{j_3} \sum_{l \in g_i | l \neq j_1, l \neq j_2, l \neq j_3} \left(x_{ljh}^{k} + x_{jlh}^{k}\right) = \left(x_{41h}^{k} + x_{14h}^{k}\right) + \left(x_{51h}^{k} + x_{15h}^{k}\right) + \dots + \left(x_{73h}^{k} + x_{37h}^{k}\right) = 0$$

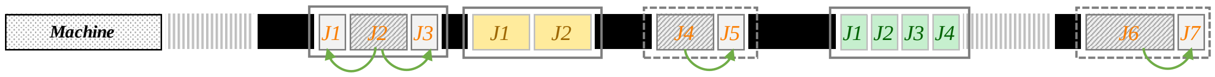
$$\sum_{j=j_1}^{j_3} \sum_{j'=j+1}^{j_3} \left(x_{jj'h}^{k} + x_{j'jh}^{k}\right) = \left(x_{21h}^{k} + x_{12h}^{k}\right) + \left(x_{31h}^{k} + x_{13h}^{k}\right) + \left(x_{32h}^{k} + x_{23h}^{k}\right) = 2$$

The number of ECs and ICs related to selected jobs, i.e., $\{1, 2, 3\} \in Q_{ih}^{k}$, are 0 and 2, respectively. Consequently,

$$\begin{cases} M\left(Y_{123}^{ihk}\right) \geq 2 \\ M\left(Y_{123}^{ihk} - 1\right) \leq 2 - \varepsilon\left(Y_{123}^{ihk}\right) \end{cases} \Longrightarrow Y_{123}^{ihk} = 1 \text{ and equation (4.37) is satisfied as follows:}$$

$$(0) + 2(2) \geq 2(1)(3 - 1) \Rightarrow 4 \geq 4$$

Scenario 2: consider the following selected jobs, i.e., $\{2, 4, 6\} \in Q_{ih}^{k}$:



$$\sum_{j=j_1}^{j_3} \sum_{l \in g_i | l \neq j_1, l \neq j_2, l \neq j_3} \left(x_{ljh}^k + x_{jlh}^k\right) = \left(x_{12h}^k + x_{21h}^k\right) + \left(x_{32h}^k + x_{23h}^k\right) + \cdots + \left(x_{76h}^k + x_{67h}^k\right) = 4$$

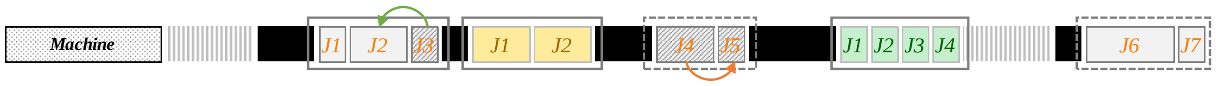
$$\sum_{j=j_1}^{j_3} \sum_{j'=j+1}^{j_3} \left(x_{jj'h}^k + x_{j'jh}^k\right) = \left(x_{42h}^k + x_{24h}^k\right) + \left(x_{62h}^k + x_{26h}^k\right) + \left(x_{64h}^k + x_{46h}^k\right) = 0$$

The ECs and ICs for selected jobs, i.e., $\{2, 4, 6\} \in Q_{ih}^k$, are 4 and 0, respectively. Consequently,

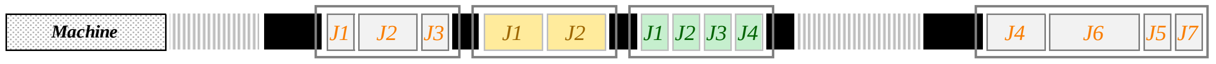
$$\begin{cases} M\left(Y_{246}^{ihk}\right) \geq 0 \\ M\left(Y_{246}^{ihk} - 1\right) \leq 0 - \varepsilon(Y_{246}^{ih}) \end{cases} \Longrightarrow Y_{246}^{ihk} = 0 \text{ and equation (4.37) is satisfied as follows:}$$

$$(4) + 2(0) \geq 2(0)(3 - 1) \Rightarrow 4 \geq 0$$

Scenario 3: consider the following selected jobs, i.e., $\{3, 4, 5\} \in Q_{ih}^k$:



Likewise, the ECs and ICs for selected jobs, i.e., $\{3, 4, 5\} \in Q_{ih}^k$, are equal to 1 each. Consequently, $Y_{345}^{ihk} = 1$ and equation (4.37) is not satisfied since $(1) + 2(1) \ngeq 2(1)(3 - 1)$. Therefore, the jobs must be re-arranged to satisfy LB_{ih}^k as in the following sequence:



In conclusion, constraints (4.37) through (4.39) guarantee that any developed batch of group i on machine h satisfies LB_{ih}^k; otherwise this batch will not be developed on machine h in stage k.

4.1.4. RMILP

It is essential to develop a robust measure, i.e., a lower bound, to evaluate the performance of meta-heuristic algorithms developed for the research problem. Therefore, the relaxed MILP model is developed to obtain an optimal solution or a good quality lower bound in affordable computational time, particularly for large-size problems. The RMILP model reduces the large solution space considerably by eliminating the batching phase from the previous MILP models (Appendix A). In other words, unlike the previous MILP models focusing on all combinations between batch compositions of groups, the RMILP model focuses on only one combination. The RMILP model can be developed based on any of the proposed MILP models. In this research, the RMILP model is a relaxed version of the MILP1 model, which is developed based on the precedence constraints between each pair of jobs on machines. Appendix B shows a relaxed version of the

MILP2 model, as another example, which is developed based on the position concept within machines to determine the job sequence. The sets, subsets, indices, parameters, decision variables, and mathematical formulation for the RMILP are presented next.

Sets and Indices

G	Set of groups, indexed by i, p	$G = \{1,2,\dots,g\}$
G_i	Set of jobs of group i, indexed by j, q	$G_i = \{1,2,\dots,n_i\}$
K	Set of stages, indexed by k	$K = \{1,2,\dots,m\}$
V^k	Set of machines in stage k, indexed by h	$V^k = \{1,2,\dots,v^k\}$

Subsets

I^k	Subset of groups, which must be processed in stage k	$I^k \subset G$
J_i^k	Subset of jobs of group i, which must be processed in stage k	$J_i^k \subset G_i$
V_{ij}^k	Subset of machines in stage k, which can process job j of group i	$V_{ij}^k \subset V^k$
K_{ij}	Subset of stages in an ascending order, which must be visited by job j of group i	$K_{ij} \subset K$

Parameters

g	Number of groups
n_i	Number of jobs of group i
n_i^k	Number of jobs of group i, which must be processed in stage k
m	Number of stages
v^k	Number of machines in stage k
m_{ij}	Number of stages, which must be visited by job j of group i
$st_{ij(l)}$	l^{th} stage among subset K_{ij}
t_{ijh}^k	Run time of job j of group i on machine h in stage k
S_{pih}^k	Required setup time to process a batch of group i on machine h in stage k if a batch of group p is the preceding batch ($p = 0$ refers to the reference batch)
d_{ij}	Due date of job j of group i
r_{ij}	Release time of job j of group i
w_{ij}	Weight of job j of group i
a_h^k	Availability time of machine h in stage k
α	Weight attributed to the producer
β	Weight attributed to the customer

LB_{ih}^{k}	Desired lower bound for the minimum number of jobs assigned to any batch of group i on machine h in stage k

Decision variables

X_{ij}^{k}	The completion time of job j of group i in stage k
T_{ij}	The tardiness of job j of group i
Z_{ijh}^{k}	1 if job j of group i is assigned to machine h in stage k; 0 otherwise
$A_{pj'ij}^{k}$	1 if job j of group i is processed after job j' of group p in stage k; 0 otherwise

Mathematical formulation

$$Min\, Z = \alpha \sum_{i\in G}\sum_{j\in g_i} w_{ij} X_{ij}^{st_{ij(m_{ij})}} + \beta \sum_{i\in G}\sum_{j\in g_i} w_{ij} T_{ij} \tag{4.46}$$

The objective function (4.46) is to *simultaneously* minimize the total weighted completion time and total weighted tardiness.

$$\sum_{h\in v_{ij}^{k}} Z_{ijh}^{k} = 1$$
$$i \in I^{k};\ j \in J_i^{k};\ k \in K; \tag{4.47}$$

$$X_{ij}^{k} + M\left(1 - A_{pj'ij}^{k}\right) + M\left(1 - Z_{ijh}^{k}\right) + M\left(1 - Z_{pj'h}^{k}\right) \geq X_{pj'}^{k} + S_{pih}^{k} + t_{ijh}^{k}$$
$$i,p \in I^{k}(p \leq i; p = i \rightarrow t < s);\ j \in J_i^{k};\ j' \in J_p^{k};\ h \in v_{ij}^{k} \cap v_{pj'}^{k};\ k \in K;\ M\colon large\ number; \tag{4.48}$$

$$X_{pj'}^{k} + M\left(A_{pj'ij}^{k}\right) + M\left(1 - Z_{ijh}^{k}\right) + M\left(1 - Z_{pj'h}^{k}\right) \geq X_{ij}^{k} + S_{iph}^{k} + t_{pj'h}^{k}$$
$$i,p \in I^{k}(p \leq i; p = i \rightarrow t < s);\ j \in J_p^{k};\ j' \in J_p^{k};\ h \in v_{ij}^{k} \cap v_{pj'}^{k};\ k \in K;\ M\colon large\ number; \tag{4.49}$$

Constraint (4.47), known as assignment constraint, is incorporated into the model to determine the optimal job assignment on machines in each stage, while set of constraints (4.48) and (4.49), known as scheduling constraint set, determine the optimal job sequence on machines with regard to the assignment constraint.

$$X_{ij}^{k} \geq \sum_{h\in v_{ij}^{k}} \left(a_h^{k} + S_{0ih}^{k} + t_{ijh}^{k}\right) Z_{ijh}^{k} \tag{4.50}$$

$$i \in I^k;\ j \in J_i^k; k \in K;$$

$$X_{ij}^{st_{ij(1)}} \geq r_{ij} + \sum_{h \in v_{ij}^{st_{ij(1)}}} t_{ijh}^{st_{ij(1)}} \left(Z_{ijh}^{st_{ij(1)}}\right) \tag{4.51}$$

$$i \in G; j \in G_i;$$

Constraint (4.50) together with constraint (4.51) account for dynamic machine availability and dynamic job release time, respectively.

$$X_{ij}^{st_{ij(l)}} - X_{ij}^{st_{ij(l-1)}} \geq \sum_{h \in v_{ij}^{st_{ij(r)}}} t_{ijh}^{st_{ij(l)}} \left(Z_{ish}^{st_{ij(l)}}\right) \tag{4.52}$$

$$i \in G; j \in G_i;\ l \in \{2,3,\dots,m_{ij}\};$$

The linking constraint (4.52) ensures the connection between completion times of a job related to each of two sequential stages, where the job had operations.

$$T_{ij} \geq X_{ij}^{st_{ij(m_{ij})}} - d_{ij} \tag{4.53}$$

$$i \in G; j \in G_i;$$

Constraint (4.53) is applied to find the tardiness of each job.

$$X_{ij}^k, T_{ij} \geq 0;$$

$$Z_{ijh}^k \in \{0,1\}; A_{pj'ij}^k \in \{0,1\}\ (p \leq i; p = i \rightarrow t < s); \tag{4.54}$$

$$i \in G; j \in G_i; k \in K; i,p \in I^k;\ j \in J_i^k;\ j' \in J_p^k;\ h \in v_{ij}^k;\ M\text{: large number}.$$

Finally, constraint (4.54) defines the variables used. It is worth noting that the number of decision variables and constraints of the RMILP model are considerably less than those related to the previous MILP models.

4.1.4.1. Functionality of the RMILP model

Although the RMILP model is still strongly NP-hard (Du and Leung 1990), it drastically reduces the gap between the upper and lower bounds of solutions compared to this gap for the MILP models, especially for large-size problems. It is worth noting that a lower bounding mechanism or a branch-and-price algorithm can also present a good quality lower bound compared to the RMILP model, but the computational time is high. As a result, during the same computational time limit, the optimal solution of the RMILP model is equal to the optimal solution of the MILP models, provided the RMILP model does not violate any LB_{ih}^{k}; otherwise, the optimal solution and lower bound identified for the RMILP and any of the MILP models, respectively, determine the best lower bound for the MILP model. Likewise, the lower bounds identified for both RMILP and any of the MILP models determine the best lower bound for the MILP model, when the optimal solution for the RMILP model is not attainable during the computational time limit. In this case, the optimal solution/lower bound identified for both RMILP and any of the MILP models determine the best lower bound for the problem to evaluate the performance of meta-heuristic algorithms. Thus, given the computational time limit (CT_{limit}), there is a possibility of obtaining an optimal solution/a good quality lower bound of any of the MILP models by the RMILP model, particularly for medium- and large-size problems. The pseudo-code of this mechanism is shown in Table 1.

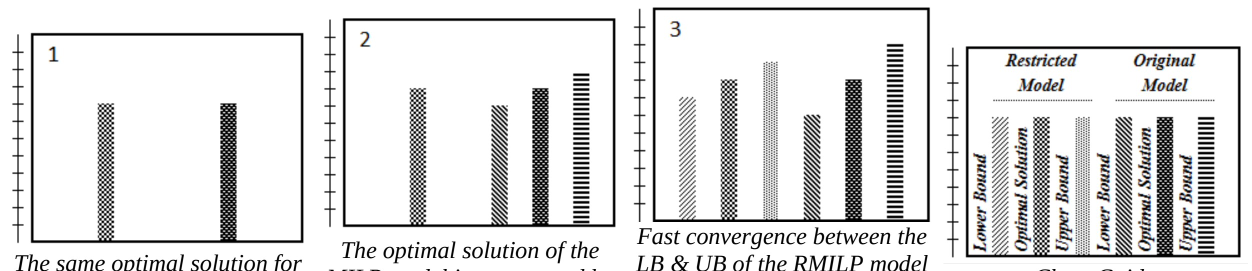
Table 1. Pseudo-code for $LB_{selective}$

```
     Result of the MILP models
1:   **Input**: $MILP_{RD}$ & $MILP_{OL}$
2:   **Output**: $LB_{selective}$ & $UB_{selective}$
3:   Solve $MILP_{RD}$ during $CT_{limit}$
4:       **if** $Opt_{RD}$ is attainable during $CT_{limit}$ **then**
5:           **if** $Opt_{RD}$ does not violate any $LB_{ih}^{k}$ **then**
6:               $LB_{selective} = UB_{selective} = Opt_{RD}$
7:           **else**
8:               Solve $MILP_{OL}$ during $CT_{limit}$
9:                   **if** $Opt_{OL}$ is attainable during $CT_{limit}$ **then**
10:                      $LB_{selective} = UB_{selective} = Opt_{OL}$
11:                  **else**
12:                      $LB_{selective} = \max\{Opt_{RD}, LB_{OL}\}$
                         $UB_{selective} = UB_{OL}$
13:                  **end if**
15:          **end if**
15:      **else**
16:          **if** $Opt_{OL}$ is attainable during $CT_{limit}$ **then**
17:              $LB_{selective} = UB_{selective} = Opt_{OL}$
18:          **else**
19:              $LB_{selective} = \max\{LB_{RD}, LB_{OL}\}$
                 $UB_{selective} = UB_{OL}$
20:          **end if**
21:      **end if**
22:  **return** $LB_{selective}$ & $UB_{selective}$
```

$MILP_{OL}$ and $MILP_{RD}$ in Table 1 represent the original and relaxed MILP models, respectively, while Opt_{OL}/Opt_{RD} and LB_{OL}/LB_{RD} represent the optimal solution and lower bound of $MILP_{OL}/MILP_{RD}$, respectively. The original MILP model refers to the MILP1, MILP2, and MILP3 models. The result of Opt_{OL}/LB_{OL} and/or Opt_{RD}/LB_{RD} determine the best lower bound of $MILP_{OL}$, i.e., $LB_{selective}$, while the result of Opt_{RD} and/or Opt_{OL}/UB_{OL} determine the best upper bound of $MILP_{OL}$, i.e., $UB_{selective}$. UB_{OL} is the upper bound of $MILP_{OL}$.

In summary, the optimal solution of the RMILP model is certainly equal to the optimal solution for any of the MILP models, when $LB_{ih}^{k} = 1, \forall i \in I^{k}; h \in V^{k}; k \in M$; otherwise, the optimal solution of the RMILP model might be either the optimal solution or the lower bound for the MILP models, when $LB_{ih}^{k} > 1, \exists i \in I^{k}, h \in V^{k}; k \in M$. It is worth noting that despite being a desired lower bound(s) on batch sizes, the optimal solutions of both MILP and any of the RMILP models are equal, when the optimal solution of the RMILP model does not violate any desired lower bounds on batch sizes. Therefore, the deviation between the optimal solutions of both MILP and any of the RMILP models indicates at least one of the desired lower bounds on batch sizes is violated by the RMILP model.

This superiority in the quality of lower bounds is intensified when most of the groups have the desired lower bounds on batch sizes close to 1 (i.e., $LB_{ih}^{k} \rightarrow 1$). This being the case, the evaluation of meta-heuristic algorithms will be performed by the outcome of optimal solutions/lower bounds of the RMILP model and lower bounds of any of the MILP models (pseudo-code presented in Table 1). With respect to no/any violation on desired lower bounds, all possible combinations between the optimal solutions and lower bounds of both RMILP and any of the MILP models are depicted in Figures 7 and 8. With the help of lines shown for the optimal solutions, lower bounds, and upper bounds of both MILP and RMILP models in Figures 7 and 8, the reader can interpret the relationships between the optimal solution/lower bound of the RMILP model and the lower bound of the MILP model, in all possible cases depicted in Figures 7 and 8.



The same optimal solution for all models with no violation on LB_b by the RMILP model

The optimal solution of the MILP model is guaranteed by the RMILP model

Fast convergence between the LB & UB of the RMILP model compared to the LB & UB of the MILP model

Chart Guide

Figure 7. No violation on desired lower bounds for the RMILP model

Since LB_{ih}^{k} is not violated by the RMILP model in all possible cases in Figure 7, the optimal solutions/lower bounds of the RMILP model are the optimal solutions/good quality lower bounds for the MILP model. Unlike cases shown in Figure 7, LB_{ih}^{k} is violated by the RMILP model in all cases shown in Figure 8. In this case, the outcome of optimal solutions/lower bounds of the RMILP model and lower bounds of the MILP models determine the best lower bounds for the MILP models.

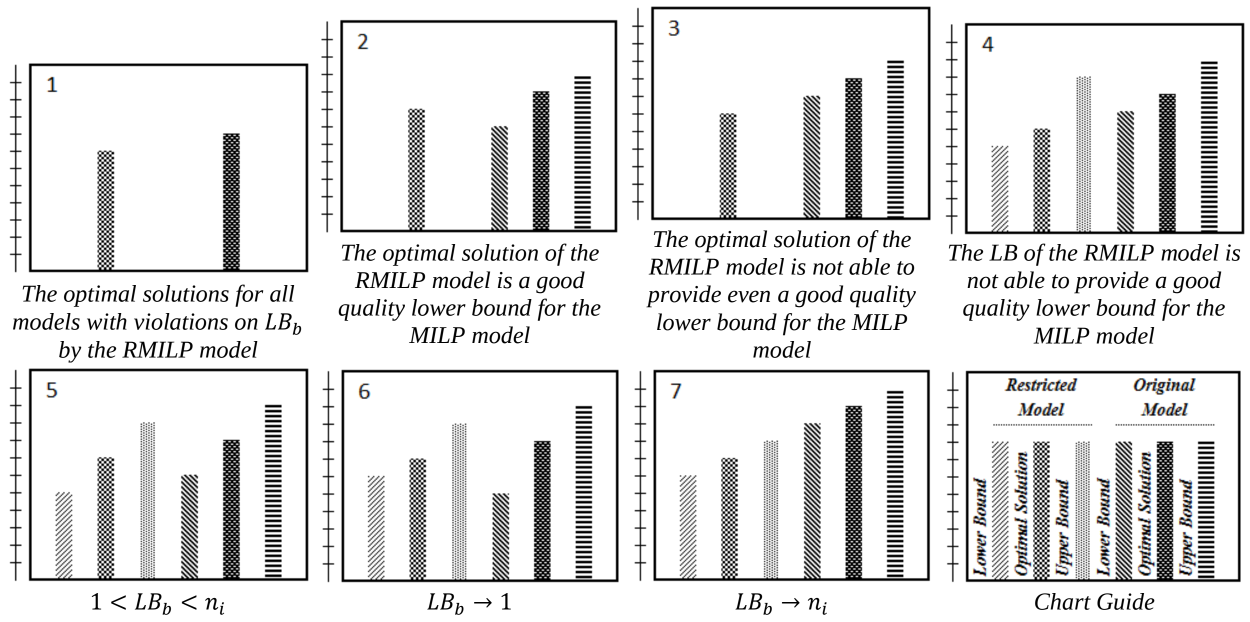


Figure 8. Violation on desired lower bounds for the RMILP model

Generally, the lower and upper bounds of the RMILP model converge faster than the ones of the original MILP model, when there is no violation on LB_{ih}^{k} by the RMILP model (case 3 in Figure 7). This fast convergence might be true for the RMILP model even if it violates LB_{ih}^{k}, when the number of violations in LB_{ih}^{k} is not significant (case 6 in Figure 8). This being the case, the RMILP model still gives a good estimate of lower bounds for any of the MILP model. Apart from this, the optimal solutions/lower bounds of the RMILP model might not provide good quality lower bounds for any of the MILP models (case 7 in Figure 8), when the number of violations in LB_{ih}^{k} is significant and/or most of the groups have the desired lower bounds on batch sizes close to n_i (i.e., $LB_{ih}^{k} \rightarrow n_i$). The reason lies in the fact that the combinations between batch compositions are decreased significantly, when $LB_{ih}^{k} \rightarrow n_i$ for most of the groups, and consequently, the lower and upper bounds of the MILP model converge faster. Therefore, the quality of the lower bound identified by any of the MILP model is better than the one identified by the RMILP model.

4.1.5. Comparison of MILP1, MILP2, MILP3, and RMILP

With the help of several test problems, the following results are obtained from a comparison between the MILP models and the RMILP model with 8 hours' time limit presented in Table 2:

- The MILP3 model develops good quality solutions for all test problems, compared to the solutions developed by the MILP1 and MILP2 models.
- The MILP3 model is capable of obtaining the optimal solution for 9 out of 19 problems, while for the rest of the problems it presents good quality lower bounds compared to the MILP1 and MILP2 models.
- Compared to the MILP1 and MILP2 models, the performance of the MILP3 model increase as the size of problems is increased.
- The performance of the MILP2 model is slightly better than the MILP1 model, particularly for large-size problems.
- Although the RMILP model is capable of obtaining the optimal solutions for 13 out of 19 problems, it presents the optimal solution of MILP models only for 6 out of 13 problems due to violation on the desired lower bounds on batch sizes.
- For all test problems, the MILP1 and RMILP models have the maximum and minimum computational times to identify optimal solutions, while the MILP2 and MILP3 models have the second and third maximum computational times, respectively.

The MILP1 and MILP2 models are developed in two integrated phases, i.e., the batching and scheduling phases. Therefore, for any combination between batch compositions determined in the batching phase, the batch sequence on machines and job sequence within batches should be determined by the precedence constraints and/or the position concept in the scheduling phase. Although both the MILP3 and RMILP models focus on transferring a batch scheduling problem formulated by the MILP1 and MILP2 to a job scheduling problem by eliminating the batching phase, they are different with respect to considering LB_b. In other words, unlike the RMILP model, the MILP3 model considers LB_b and, consequently guarantees identifying the optimal solution for the batch scheduling problem.

Our preliminary experiments revealed that the MILP3 model is capable of finding optimal solutions or good quality lower bounds in less computational time, particularly for large-size problems, compared to the MILP1 and MILP2 models. Thus, the MILP3 model along with the RMILP model have been used to determine optimal solutions/lower bounds of problems.

Table 2. Comparison between MILP models and the RMILP model

				MILP1		MILP2		MILP3		RMILP			
Problem	g	$\sum n_i$	m	OFV	LB	OFV	LB	OFV	LB	OFV	LB	Violation on LB_{ih}^{k}	Best measure to evaluate the performance of a meta-heuristic algorithm
1	5	15	2	1782.02	1260.80	1752.02	1738.02	1738.80		1682.32		Yes	1738.80
2	3	9	3	3697.54		3697.54		3697.54		3697.54		No	3697.54
3	3	9	2	956.20		956.20		956.20		914.15		Yes	956.20
4	3	9	2	1330.06	1302.98	1303.06		1303.06		1278.80		Yes	1303.06
5	3	10	3	2287.60		2287.60		2287.60		2287.60		No	2287.60
6	5	15	4	5748.70	4071.99	5696.80	4376.80	5502.69		5502.69		No	5502.69
7	3	15	5	4300.60	3688.21	4396.78	3770.12	4209.71	4180.87	4180.87		No	4180.87
8	3	9	4	4318.08	3895.44	4329.65	3945.70	4298.00		4247.63		Yes	4298.00
9	3	9	4	1579.95	1516.75	1568.39	1523.50	1551.67	1548.80	1479.95		Yes	1548.80
10	3	9	6	4808.82	3997.15	4692.70	4061.90	4495.08		4495.08		No	4495.08
11	4	15	6	4651.47	3517.29	4607.26	4167.98	4523.56		4451.47		Yes	4523.56
12	3	9	6	5787.07	4802.98	5700.26	5151.08	5502.18	5468.80	5459.97		No	5459.97
13	4	15	6	7489.88	6095.70	7729.56	7129.40	7398.70	7359.34	7264.90		Yes	7359.34
14	4	20	4	8488.69	8009.89	8463.22	8193.76	8359.83	8295.37	8129.67	7972.88	NK	8295.37
15	5	20	5	13871.80	9571.54	12595.59	11765.80	12128.43	11988.07	11983.80	11689.86	NK	11988.07
16	6	28	3	16872.90	12579.76	15822.28	11598.32	13981.20	13039.20	12873.88	12688.41	NK	13039.20
17	5	29	5	18823.80	14677.92	19030.86	13787.60	15781.49	14756.88	14987.20	14698.34	NK	14756.88
18	6	24	2	3487.56	3038.90	3308.09	3069.90	3298.90	3087.20	2989.40	2871.89	NK	3087.20
19	7	47	4	19182.90	13729.98	18760.88	14138.50	16993.43	15382.88	15879.70	15028.21	NK	15382.88

NK, OFV, and LB stand for “Not Known”, “Objective Function Value”, and “Lower Bound”, respectively.

4.2. Meta-heuristic algorithm

The problem addressed here is strongly NP-hard and, consequently, its complexity motivates investigating into the development of an algorithm, which can efficiently explore the solution space and find the best solution from among local optima. Heuristic and meta-heuristic algorithms are the main approaches used in dealing with medium- and large-size problems, which are strongly NP-hard. Apart from this, not only meta-heuristic algorithms usually show higher performance compared to heuristic algorithms due to using a mechanism to avoid getting trapped in local optima, but also their performance differs in dealing with different types and sizes of problems.

A basic local search meta-heuristic, a population-based meta-heuristic, and a combination of both have been developed for scheduling problems. A local search algorithm is an iterative algorithm so that, in each iteration, it searches for a solution in a local area in the solution space. This local area including neighbor solutions is usually the neighborhood of a particular solution called *seed*. Therefore, the search starts with an initial seed and searches the neighborhood of this solution to find a possible seed for the next iteration. The next seed is usually selected based on its quality compared to other solutions in the neighborhood. A population-based structure is an evolutionary algorithm so that, in each iteration, a population of solution is converted to another population with superior characteristics of the older population. Therefore, an iteration of a population-based structure can be considered as several consecutive iterations of a local search structure.

For the research problem, eight algorithms are considered so that two out of eight algorithms are developed in terms of a basic local search and a basic population-based meta-heuristic, while the rest of them are developed based on either a local search meta-heuristic enhanced with a population-based structure or a population-based meta-heuristic enhanced with a local search structure. The notation of the MILP1 model is used to show the equations in developed meta-heuristics. Before going into the details of the developed meta-heuristic algorithms, common implementation strategies are explained first.

4.2.1. Move interdependency

The optimal solution of HFS problems are not generally in the form of permutation sequences, but very close to these sequences. Since there is a different combination between batch compositions of all groups in each stage of HFS in batch scheduling, the permutation schedule does not hold true in the problem addressed in this research. Therefore, a solution might be represented by different batch compositions in each stage.

Since there is interdependency between positions of a job in different stages of HFS in batch scheduling, a meta-heuristic algorithm is not capable of capturing these interdependencies and, subsequently, its efficacy can be diminished. Therefore, the move interdependency (MI) is referred to the impact of either a move created on k^{th} stage or several concurrent moves created on several stages up to k^{th} stage on the job and/or batch sequence and assignment related to the following stages, i.e., $k+1,\dots,m$. In a local search structure, a move in k^{th} stage of HFS for creating an adjacent solution should be accompanied by the MI for the following stages. Since several concurrent moves related to a stage(s) of HFS might create an adjacent solution in a population-based structure, it is more critical and complicated to accompany the moves' interdependencies on the following stages, compared to a local search structure.

Therefore, after applying a move or several concurrent moves for creating an adjacent solution, this move can be accompanied with and without considering the MI for the following stages. Thus, the batch compositions corresponding to $k+1, k+2,\dots,m$ might be changed with respect to the MI in the scheduling and/or the batching phase, while they are assumed to be fixed (i.e., the same as its seed/parent) without considering the MI. Unlike group scheduling problems, due to different combinations of batch compositions for each stage of HFS in batch scheduling problems, the MI can be dealt with one of the following strategies:

Non-interdependency strategy**:** A single perturbation performed on k^{th} stage or several perturbations performed up to k^{th} stage is not accompanied by any changes on $k+1,\dots,m$ stages. This strategy evaluates a move or concurrent moves without considering the MI. Since a single perturbation or several perturbations is performed on a stage or up to a stage, respectively, while all the other following stages remain unchanged, the outcomes of this perturbation(s) is non-interdependency sequence, which clarifies the name of this strategy.

Performing any perturbation(s) on HFS environments may result in the emergence of idle times (delays) in the entire schedule. These predictable delays, which represent a potential deficiency of meta-heuristic algorithms in dealing with the HFS scheduling problems, lead the efficiency of a meta-heuristic algorithm to be drastically diminished. Apart from this, a single perturbation in a local search algorithm can change the position of multiple jobs at each perturbation. Therefore, the multiple combinations of perturbations in a population-based algorithm to generate each solution may result in a non-polynomial search algorithm, which is not desirable in dealing with strongly NP-hard problems. In order to avoid these deficiencies in dealing with the HFS batch scheduling problem, one remedy is to account for simultaneous interdependencies (partial- or complete- interdependencies). In other words, while changing the position of a job(s) and/or batch(s) in any stage(s), batch composition of groups as well as positions of jobs in the following stages should be checked and accordingly changed to avoid the possible delays.

Partial-interdependency strategy**:** This strategy retains the same batch compositions and job order within batches from k^{th} stage, i.e, $k = 1,2, \dots, m - i - 1$, up to the last stage, for each of two consecutive stages, i.e, $(k + i)^{th}$ & $(k + i + 1)^{th}$ stages, where $i = 0,1,2, \dots, m - k - 1$, after accounting for the adjustment step (in the following section 4.2.3) for the first stage of each two consecutive stages. The sequence and assignment of batches on machines corresponding to each stage of $k + 1, \dots, m$ stages are determined based upon both the earliest available machine times and earliest batch availability times. The batch availability time in $(k + i + 1)^{th}$ stage of each of two consecutive stages is determined in terms of the completion time of its first job in $(k + i)^{th}$ stage.

Complete-interdependency strategy**:** Unlike the partial-interdependency strategy, this strategy does not keep the same batch composition for each of two consecutive stages. This strategy determines only job orders on machines (not within batches) for $k + 1, \dots, m$ stages, based upon both the earliest available machine times and earliest job availability times, regardless of which batch a job is assigned to before. This strategy applied for each two consecutive stages of $k, k + 1, \dots, m$, determines *simultaneously* the batch compositions, batch sequence and assignment on machines, and job sequence within batches. In other words, for each of two consecutive stages, the first available job of a group in the prior stage is assigned to the first available machine on the following stage as a batch. The other available jobs of the same group in the prior stage must be assigned to the same batch on the following stage based on their availability times, until the desired lower bound for this batch is satisfied. Then, for the rest of jobs belonging to this group, the same process follows until all jobs are assigned to a machine(s) as batches.

As much as possible, the machine eligibilities for a job/batch assignment to a machine(s) and LB_{ih}^{k} for job assignment must be considered in both partial- and complete-interdependency strategies. Ties will be broken in favor of the smallest machine and/or batch and/or job index in both procedures.

It is obvious that an effective move(s) cannot be created without considering the MI, due to emergence of idle times in the entire schedule. Apart from this, with respect to the MI, partial-interdependency strategy takes the flexibility of batch scheduling for stages $k + 1, k + 2, \dots, m - 1$ and m, because it behaves as in group scheduling and follows the GTAs (the same batch composition for these stages. Furthermore, complete-interdependency strategy increases the number of setups, because the job assignments are based on only job availability times. Therefore, complete-interdependency strategy results in not only its objective function value being worse, but also its schedule cost to be increased. These arguments led us to probe into the properties of the stage-based interdependency strategy in the HFS batch scheduling problem, described next.

***Stage-based interdependency strategy*:** Creating idle times, taking the flexibility of batch scheduling, and increasing the number of setups, are the potential deficiencies of non-, partial-, and complete-interdependency strategy, respectively. These deficiencies lead to the reduction in the efficiency of the local search and population-based algorithms. Stage-based interdependency strategy gradually determines the best job and batch sequences as well as batch compositions in each stage. Considering the MI, this strategy is defined with the help of the following statement:

Regardless of setting batch compositions, machine assignments, batch sequences on machines, and job sequences within batches which are determined up to $(\boldsymbol{k-1})^{\boldsymbol{th}}$ stage, the best of their setting corresponding to $\boldsymbol{k^{th}}$ stage is determined with the help of stage-based release time and stage-based due date.

As a result of this strategy, the HFS problem is decomposed into m sub-problems ($SP_k, k \in m$), where each sub-problem is a batch scheduling problem on either a single machine or a set of unrelated-parallel machines. The sub-problems link together with the help of stage-based release time (similar to constraints (4.9) and (4.13) in section 4.1.1), while the approximate impact, known as load ratio (LR_{ij}^k), of tardiness of jobs is considered with the help of stage-based due date in each stage.

The stage-based release time of a job is equivalent to the completion time of the job in the immediately preceding stage. The most important point in developing the stage-based interdependency strategy is to follow the bi-criteria objective function in each stage. Thus, the stage-based due date needs to be considered in each stage due to the tardiness criterion in the bi-criteria objective function. The stage-based due date, which is based upon modified due dates, is obtained with the help of the load ratio of real due dates, so that the stage-based due date will be equivalent to the real due date in the last stage. Therefore, the stage-based due date considers customers' interests in all stages of batch scheduling. The stage-based release time ($\hat{r}_{ij}^k$) and stage-based due date ($\hat{d}_{ij}^k$) of job j of group i in k^{th} stage are determined by the following equations:

$$\hat{r}_{ij}^k = C_{ij}^{k-1} \tag{4.55}$$

$$\hat{d}_{ij}^k = LR_{ij}^k * d_{ij} \tag{4.56}$$

where C_{ij}^k is the completion time of job j of group i in k^{th} stage. Load ratios are obtained as follows:

$$LR_{ij}^{k} = \left({\left(\sum_{l=1}^{k} \bar{S}_{ij}^{l} + \bar{t}_{ij}^{l} \right)} \Big/ {\left(\sum_{l \in K} \bar{S}_{ij}^{l} + \bar{t}_{ij}^{l} \right)} \right) \tag{4.57}$$

$$\bar{S}_{ij}^{k} = \left({\left(\sum_{h \in V_{ij}^{k}} \bar{S}_{ih}^{k} \right)} \Big/ {\xi_{ij}^{k}} \right) \tag{4.58}$$

$$\bar{t}_{ij}^{k} = \left({\left(\sum_{h \in V_{ij}^{k}} t_{ijh}^{k} \right)} \Big/ {\left(\sum_{h \in V_{ij}^{k}} \partial_{ih}^{k} \right)} \right) \tag{4.59}$$

$$\bar{S}_{ih}^{k} = \left({\left(\sum_{p \in I^{k}+\{0\}} S_{pih}^{k} \right)} \Big/ {g} \right) \tag{4.60}$$

ξ_{ij}^{k} indicates the total number of machines, which can process job j of group i in k^{th} stage. ∂_{ih}^{k} indicates the total number of jobs in group i, which can be processed by machine h in k^{th} stage. The stage-based interdependency strategy is shown in Figure 9.

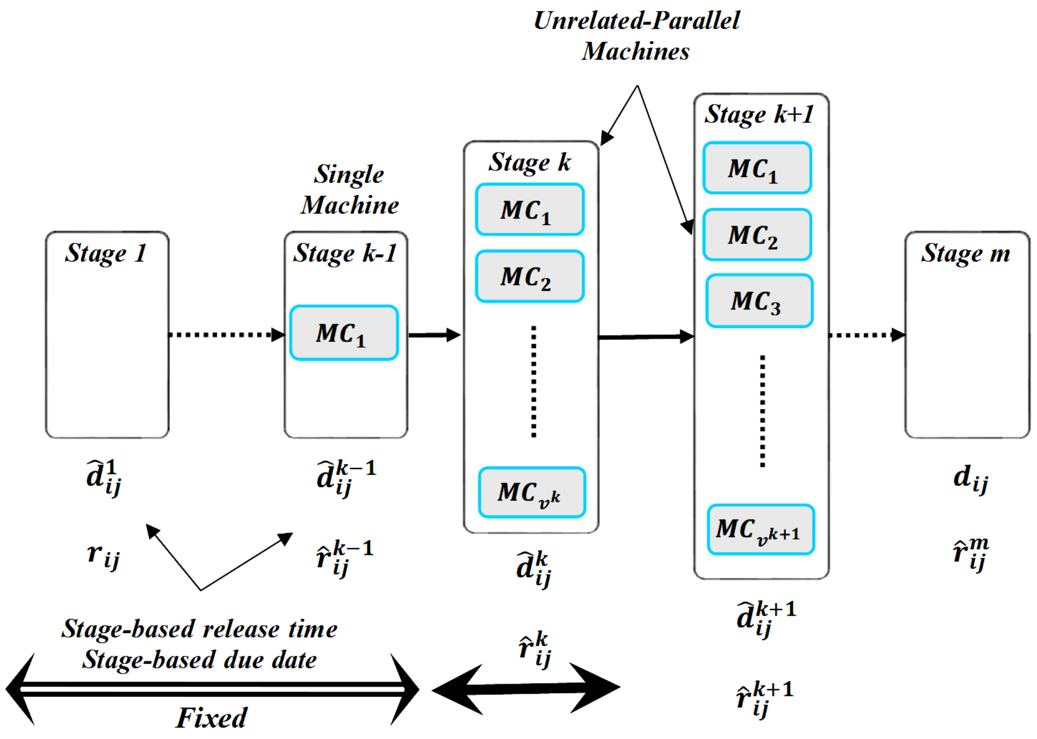


Figure 9. The stage-based improvement procedure along with the three-level TS algorithm

In the first and last sub-problem, i.e., (SP_1) and (SP_m), $\hat{r}_{ij}^{1} = r_{ij}$ and $\hat{d}_{ij}^{m} = d_{ij}$, respectively. The sign ⇔↔ in Figure 9 indicates that the best schedule in k^{th} stage is determined only with the help of the SBRT, SBDD, and developed IS (in the following section 4.2.2) in k^{th} stage, independent of the best fixed schedules up to $(k-1)^{th}$ stage ($k = 2, \dots, m$). In the stage-based interdependency strategy, a partial solution represents the sequence from the first stage up to k^{th} stage ($k = 1, \dots, m-1$). A complete solution is obtained for all stages of the stage-based interdependency strategy, only when the search algorithm reaches the last stage.

The preliminary results show the superior performance of the stage-based interdependency strategy on local search and population-based structures for batch scheduling in HFS. Although the other interdependency strategies might be capable of obtaining better results compared to the stage-based interdependency strategy, a paired *t*-test performed to compare different strategies at a significance level of 5% for each comparison shows there is a significant difference between the stage-based interdependency strategy and the other strategies. Therefore, only this strategy is accompanied by all meta-heuristic algorithms in this research.

4.2.2. Initial solution finding mechanism

An initial solution finding mechanism is applied to generate the initial solution (IS) and, consequently, the initial population (IP) of meta-heuristic algorithms. From a combinational optimization perspective, a randomly generated solution frequently is of poor quality and thus is too costly or impossible to improve that to an optima. Therefore, in order to be efficient in finding good quality IS, an IS finding mechanism is generated since the quality of the final solution is sensitive to the IS (Logendran and Subur 2004). Also, the computational time is remarkably reduced when a feasible, high quality IS is used, instead of using a random solution, which might be infeasible.

Considering that the processing time is a combination of setup and run times, in a single machine job scheduling problem, the weighted shortest processing time (WSPT) and weighted earliest due date (WEDD) rules can determine the optimal/near optimal solutions for the objective functions of total weighted completion time and total weighted tardiness of jobs, respectively. Since the objective function of the proposed problem is a linear combination of two mentioned objective functions, we conjecture that developing the IS finding mechanisms, inspired by WSPT and WEDD heuristics, leads to an effective IS for the HFS scheduling problem. Since the bi-criteria objective function is considered in this research, a producer's sequence (PS) and customers' sequence (CS) can be generated (Bozorgirad and Logendran 2013) by applying modified WSPT and WEDD rules, respectively. Then, the PS and CS are combined to determine the final sequence by considering the normalization of their positional values ($\alpha.PS + \beta.CS$).

The IS is gradually determined, stage by stage, according to the progress of the stage-based interdependency strategy. The stepwise IS finding mechanism as well as finding the PS and CS are described as follows:

- **Step 0:** Consider $k = 1$.
- **Step 1:** Calculate $\hat{r}_{ij}^{k}$ and $\hat{d}_{ij}^{k}$.
- **Step 2:** Determine the group sequence (G_{SQ}) on each machine by non-decreasing sorting of $\left(\alpha.\left[PS_{ih}^{k}\right] + \beta.\left[CS_{ih}^{k}\right]\right), \forall i \in I^{k}$. $\left[PS_{ih}^{k}\right]$ and $\left[CS_{ih}^{k}\right]$ are the positional values of group i in the G_{SQ} of machine h in k^{th} stage, which are obtained by non-decreasing sorting of the following equations on machine h:

$$\left[PS_{ih}^{k}\right] \longrightarrow \left(\bar{t}_{ih}^{k} + S_{ih}^{k} + \bar{r}_{ih}^{k}\right) \tag{4.61}$$

$$\left[CS_{ih}^{k}\right] \longrightarrow \left(\bar{d}_{ih}^{k} + \bar{r}_{ih}^{k}\right) \tag{4.62}$$

  where,

$$S_{ih}^{k} = \min_{p \in I^{k} + \{0\} - \{i\}} S_{pih}^{k} \tag{4.63}$$

$$\bar{t}_{ih}^{k} = \left(\left(\sum_{j \in J_i^k} \left(t_{ijh}^{k} \gamma_{ijh}^{k} \right) \right) \Big/ \partial_{ih}^{k} \right) \tag{4.64}$$

$$\bar{r}_{ih}^{k} = \left(\left(\sum_{j \in J_i^k} \left(\hat{r}_{ij}^{k} \gamma_{ijh}^{k} \right) \right) \Big/ \partial_{ih}^{k} \right) \tag{4.65}$$

$$\bar{d}_{ih}^{k} = \left(\left(\sum_{j \in J_i^k} \left(\hat{d}_{ij}^{k} \gamma_{ijh}^{k} \right) \right) \Big/ \partial_{ih}^{k} \right) \tag{4.66}$$

- **Step 3:** Determine the batch sequence (B_{SQ}) on each machine according to the G_{SQ} on each machine, so that a permissible job(s) belonging to each group will be assigned to the earliest available machine as a batch. The permissible job is a job which does not violate the machine eligibility for processing. LB_{ih}^{k} and the machine eligibilities should be considered, as much as possible.
- **Step 4:** Apply first the refinement step and then the adjustment step for created IS.

- **Step 5:** Determine the job sequence (J_{SQ}) within each batch belonging to the B_{SQ} by non-decreasing sorting of $(\alpha.[PS_{ij}^k] + \beta.[CS_{ij}^k]), \forall i \in I^k \; \& \; j \in J_i^k | t_{ijh}^k \neq \infty$. $[PS_{ij}^k]$ and $[CS_{ij}^k]$ are the positional values of job j of group i assigned to a batch in the B_{SQ} of machine h in k^{th} stage, which are obtained by non-decreasing sorting of the following equations on machine h:

$$[PS_{ij}^k] \longrightarrow \left({(t_{ijh}^k + \hat{r}_{ij}^k)}/{w_{ij}} \right) \tag{4.67}$$

$$[CS_{ij}^k] \longrightarrow \left({(\hat{d}_{ij}^k + \hat{r}_{ij}^k)}/{w_{ij}} \right) \tag{4.68}$$

- **Step 6:** Determine the best sequence of jobs on each machine as batches in k^{th} stage with the help of a meta-heuristic algorithm.
- **Step 7:** Set $k = k + 1$ and go to step 1 until $k = m + 1$.

γ_{ijh}^k is equal to one when job j of group i can be processed by machine h in k^{th} stage.

4.2.3. Refinement step and adjustment step

After developing an initial solution/population and generating neighbor solutions relating to both local search and population-based structures, two steps might be needed: The *refinement step*, which is implemented for modifying the batch compositions of groups, batch assignment and sequencing on a machine(s), and job sequencing within some batches, after any perturbation(s) on a seed. For example, after a perturbation on a seed, a batch of group i is divided into two batches and, consequently, the batch composition of group i must be changed. Second, the *adjustment step*, which is implemented for the solutions which violate the machine eligibilities and/or LB_{ih}^k. If a solution does not meet the machine eligibility(s) and/or LB_{ih}^k, some job assignment must be changed based on the following procedure:

- **Step 1:** identify sets of jobs ($List_J$) and batches ($List_S$) separately, which violate the machine eligibilities and/or LB_{ih}^k, respectively.
- **Step 2:** $\forall j \in List_J$, assign job j to the best permissible batch $s, s \in List_S$, and revise $List_S$; otherwise assign job j to the best permissible batch $s, s \notin List_S$. The permissible batches are batches of group i which can include job j, while the best permissible batch is the one with the minimum processing time.

- **Step 3:** $\forall s \in List_S$, and for each job j of batch s, assign each job to the best permissible batch $s', s' \in (List_S - s)$, and revise $List_S$; otherwise assign this job to the best permissible batch $s', s' \notin List_S$.
- **Step 4:** apply the refinement step.

The position of assigned job j within current jobs of the new batch is the same as its current position in the current batch. It is worth noting that merging a batch which violates LB_{ih}^{k} with other possible batches of the same group, instead of using the adjustment step, might reduce the performance of the search algorithm. In the following four sections, meta-heuristic algorithms are explained in details.

4.2.4. Tabu Search

Tabu Search, a local search algorithm introduced by Glover (1986), has the ability to move through different local optima in the solution space with the help of its memory function. TS is an effective iterative meta-heuristic algorithm, which is guided by tailored neighborhood structures. TS applies tabu list as the Short-Term Memory (STM) function in order not to be trapped in a local optimum. The STM prohibits the search from choosing some of the previously identified neighborhoods to avoid being trapped in local optima. Apart from the STM, the Long-Term Memory (LTM) can be typically applied at each level of the search algorithm to explore more by intensifying and diversifying the search based upon maximum frequency (LTM-MAX) and minimum frequency (LTM-MIN), respectively. This can result in improving the quality of the objective function value. The LTM-MAX intensifies the search into a region that has been explored before more frequently, while the LTM-MIN diversifies the search into a region that has not been explored before. In the following, notations of TS-based algorithm as well as its algorithmic structure are briefly presented, and then the algorithm is completely presented with respect to solution representation, neighborhood finding mechanisms, tabu list structures, algorithmic steps, and finally the level moving in TS.

4.2.4.1. Components of Tabu Search

Initial solution (IS): TS needs an IS to trigger the search into the solution space.

Neighborhood: a local search-based algorithm searches between the neighbor solutions of each solution for possible improvements. The neighbor solution of each solution is obtained by perturbing different elements of the solution. The types and number of these perturbations are dependent on the structure and the size of the problem, respectively. A local search-based algorithm might require an excessively large computation time for solving large-size instances of the problem, if the number of perturbations is polynomial with respect to the size of the problem.

Candidate: the seed in each iteration of TS is called a candidate, referred to the potentiality of the solution for being a local or global optimum. A next candidate in each iteration is usually the best solution between neighbor solutions of the current candidate with respect to tabu list and candidate list. In other words, TS follows a mechanism that prohibits the algorithm from always selecting the best neighbor solution as a candidate to avoid getting trapped in local optima.

Candidate list (CL): the CL records each selected candidate found during the search of each iteration of the algorithm. The new entry into the CL is not necessarily better than its current members due to the existence of tabu solutions. After each insertion into the CL, the inserted candidate will be compared to the previous candidate. If the objective function value has improved then the current candidate will be assigned a star (*), which indicates it is capable of becoming the next local optima. If the objective function value has not improved and the previous entry into the CL already has a star, it will be assigned another star (**) and entered into the index list.

Index list (IL): the IL, a subset of the CL, records all local optima found during the search. A local optima in the IL is a member of the CL, which is better than both its preceding and succeeding entries into CL. A certain pre-defined number of local optima in the IL, commonly referred to as index list size (ILS), is one of the criteria used to stop the search.

Tabu list (TL): the TL is used to avoid the cyclical problem by forbidding certain perturbations (moves), which are called tabu. Therefore, in each iteration, the perturbation, which led to an entry into the CL, enters into the TL and remains there for a certain pre-defined number of iterations, which is known as tabu list size (TLS). The TL together with the TLS performs the role of what is referred to as the STM. Since the TL is a “first in first out” list, whenever the TL is filled to its capacity, the move which was forbidden for the longest duration (the oldest move in the TL) is replaced with the most recent move in the TL. A neighbor solution, obtained from tabu moves, cannot be considered as the next candidate unless it leads to the best solution found so far, i.e., better than the aspiration level defined so far. In this case, the move will re-enter into the TL.

Aspiration level (AL): the AL is the best solution found so far by the search. As mentioned, whenever a tabu move results in finding a solution better than the AL, the tabu status of that move is overridden, and the solution can be selected as the next candidate.

Stopping criteria: several criteria may be chosen for stopping the search, including the maximum number of iterations without improvement (MIWOI), maximum number of the IL/local optima (MIL), or maximum amount of CPU/computation time (MCPU). When a feasible solution is added to the CL, and its value is not less than the value of the previous member of the CL, the value of iterations without improvement

(IWOI) is increased by one; otherwise this counter is reset to zero. If either of the mentioned criteria or a combination of both attains its predefined values, the search will be stopped.

4.2.4.2. Algorithmic structure

Based upon decomposition concept, a hierarchical TS implementation is appropriate for batch scheduling problems. In order to reduce the computational burden of traditional two-level TS algorithm (job and group levels) applied for group scheduling problems as well as increase its efficiency, Shahvari and Logendran (2015, 2017) proposed a three-level TS algorithm for batch scheduling problems. A three-level TS-based algorithm first divides groups into batches and then for a created combination between batch compositions of groups, the sequence of batches on machines as well as the sequence of jobs within batches are determined, i.e., integrating the batching phase into the scheduling phase. The TS-based algorithm used in this research includes three levels, which move back and forth between batching and scheduling phases. These three levels are the central level, the outside level, and the inside level. It is worth noting that the job sequence within the same batch and the job assignment within different batches performed by the job level of a two-level TS algorithm are divided into the inside and central levels of the three-level TS algorithm, respectively. The outside level of the three-level TS algorithm, which determines the batch sequence on machines, is similar to the group level of a two-level TS algorithm. The three-level TS-based algorithm is shown in Figure 10. As it is shown, the batching phase includes splitting and merging batches, while the scheduling phase includes batch sequencing and job sequencing.

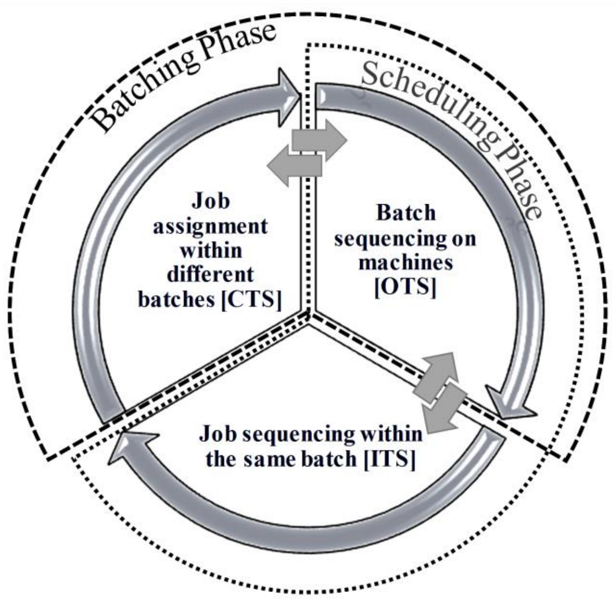

Batching Phase
Scheduling Phase
Job assignment within different batches [CTS]
Batch sequencing on machines [OTS]
Job sequencing within the same batch [ITS]

Figure 10. The two-phase solution procedure at three levels

The central tabu search (CTS) performed by job assignment within different batches, determines the batch combinations. The outside tabu search (OTS) performed by batch sequencing on machines, determines the batch sequence and assignment on each machine based upon the batch combinations, which is identified in the CTS. It is worth noting that in the batching phase, batches can be either split or merged together in the CTS, while they can only be merged in the OTS. Finally, the inside tabu search (ITS) performed by job sequencing within the same batch, determines the job sequence within each batch based upon the batch assignment and sequence, which is identified in the OTS. The CTS, OTS, and ITS are implemented in the central, outside, and inside levels, respectively. Each level of the three-level TS-based algorithm follows the same algorithmic steps of tabu search (Glover 1986).

Due to changes in at least one characteristic of a solution, i.e., a batch composition(s), machine assignment for batches, batch order on a machine(s), and job order within a batch(es), the refinement step might be implemented when the search moves from the CTS to the OTS and also from the OTS to the ITS. This step modifies the batch composition and/or machine assignment and/or batch order and/or job order in a level moving. Level moving includes any movement of a solution between levels of the three-level TS-based algorithm (in the following section 4.2.4.7). Laguna et al. (1993) used four restarts associated with the LTM for their single machine scheduling problem. Since there is three search levels in the algorithm and because the CTS plays an important role in obtaining a better solution than both OTS and ITS in this research, based upon preliminary investigations on test problems, only three restarts are performed in the CTS with the LTM-MAX or LTM-MIN, whenever it is needed.

Apart from STM in each level of TS, LTM-MAX and LTM-MIN in the highest level of TS, i.e., CTS, specific types of diversifications are activated at the middle (OTS) and lowest level (ITS) of TS. Since the first best neighbor solution identified amongst all neighbor solutions with the same objective function value is always to be selected as a seed for the next iteration of the ITS, the location of the neighbor chosen is randomly changed in order to avoid concentrating on a particular part of the solution space. Also, the sequence of batches in the current schedule is arbitrarily changed, after several iterations without improvement in the OTS.

4.2.4.3. Solution representation

With respect to the stage-based interdependency strategy and the algorithmic structure, each solution including an IS is represented with the help of a combination of the central seed (CSD^k), outside seed (OSD^k), and inside seed (ISD^k), in each stage of HFS ($\forall\ k \in m$). The CSD^k, OSD^k, and ISD^k show the job assignment and sequence on machine(s) in the CTS, the batch assignment and sequence on machine(s) in the OTS, and the job sequence within batches in the ITS, respectively, corresponding to k^{th} stage. By recognizing that j_{sij} represents job j assigned to s^{th} batch of group i, S_{si} represents s^{th} batch of group i, and j_j represents job j, a solution related to k^{th} stage of HFS including v^k unrelated-parallel machines is represented by the following set of the CSD^k, OSD^k, and ISD^k:

$$\begin{cases} CSD^k = \left[\left\{j_{sij}\right\}_{v^1} \mid \left\{j_{sij}\right\}_{v^2} \mid \dots \mid \left\{j_{sij}\right\}_{v^k}\right] \\ OSD^k = \left[\left\{S_{si}\right\}_{v^1} \mid \left\{S_{si}\right\}_{v^2} \mid \dots \mid \left\{S_{si}\right\}_{v^k}\right] \\ ISD^k = \left[\left\{j_j\right\}_{v^1} \mid \left\{j_j\right\}_{v^2} \mid \dots \mid \left\{j_j\right\}_{v^k}\right] \end{cases}$$

$\left\{j_{sij}\right\}_{v^k}$ represents the job assignment and sequence on machine h of k^{th} stage. Hyphens in each set of jobs, i.e., $\left\{j_{sij}\right\}_{v^k}$, separate between j_{sij} belonging to the same machine and vertical lines in CSD^k separate $\left\{j_{sij}\right\}_{v^k}$ between different machines. $\{S_{si}\}_{v^k}$ represents the batch sequence on machine h of k^{th} stage. Hyphens in each set of batches, i.e., $\{S_{si}\}_{v^k}$, separate between S_{si} belonging to the same machine and vertical lines in OSD^k separate $\{S_{si}\}_{v^k}$ between different machines. $\left\{j_j\right\}_{v^k}$ represents the job sequence within batches assigned to machine h of k^{th} stage. Commas and hyphens in each set of jobs, i.e., $\left\{j_j\right\}_{v^k}$, separate sequentially jobs belonging to the same batch and jobs belonging to different batches, respectively, according to the batch sequence on the same machine in the OSD^k. Also, vertical lines in ISD^k separate $\left\{j_j\right\}_{v^k}$ between different machines.

A complete solution related to the entire stages is represented by k sets of the CSD^k, OSD^k, and ISD^k, each set related to a particular stage. An example sequence related to a stage ($k \in m$) of HFS including three unrelated-parallel machines can be represented as follows:

CSD^k [j_{113}-j_{111}-j_{121} | j_{144}-j_{142}-j_{141}-j_{143}-j_{223}-j_{222} | j_{151}-j_{153}-j_{154}-j_{212}-j_{133}-j_{131}-j_{134}-j_{135}]

OSD^k [S_{11}-S_{12} | S_{14}-S_{22} | S_{15}-S_{21}-S_{13}]

ISD^k [j_3, j_1-j_1 | j_4, j_2, j_1, j_3-j_3, j_2 | j_1, j_3, j_4-j_2-j_3, j_1, j_4, j_5]

j_2 of both group 3 and 5 are skipped in k^{th} stage. The CSD^k, OSD^k, and ISD^k are all related to each other so that the CSD^k *simultaneously* includes both the OSD^k and ISD^k. Therefore, the OSD^k and ISD^k are changed, when the CSD^k changes. Likewise, the CSD^k and ISD^k are changed, when the OSD^k changes. Also, the CSD^k and/or OSD^k are changed, when the ISD^k changes.

4.2.4.4. Neighborhood Finding Mechanisms

A combination of the insertion- and swap-related operators, considered as the neighborhood structure, is performed at each level of the TS-based algorithm. Generally, the insertion-related operator is obtained by inserting a job/batch across other jobs/batches, either on the same machine or different machines, yet one move at a time. Apart from this, the swap-related operator is obtained by exchanging the positions of two jobs/batches, either on the same machine or different machines, while maintaining the same positions for the remaining jobs/batches.

With respect to both insertion- and swap-related operators, two types of moves are implemented to determine neighbor solutions: dividing move and sequencing move. The outcome of a dividing or sequencing move might lead to merging batches, when two batches of the same group are processed one after another on the same machine.

Move type 1: dividing move

The dividing move, performed by the job assignment within different families/batches, splits families/batches and changes a batch composition(s). Subsequently, the machine assignment for batches, batch order on a machine(s), and job order within a batch(s) change. Therefore, the dividing moves implemented by the CTS, determine the best batch compositions.

Since insertion-related move in the CTS provides a more reasonable change in the current schedule compared to swap-related move, moves proposed in the CTS are mostly based upon insertion techniques due to reduction in the computational burden of the search algorithm. Therefore, the neighbor solutions are obtained by inserting the jobs, yet one move at a time, between jobs across other batches, both on the same machine and different machines. Performing an insert move in the CTS can also effectively split and/or merge batches. Such divisions and merging in the CTS may directly affect the objective function value more than any move in the OTS and ITS. The central moves which violate desired lower bounds for newly formed and moved batches must be forbidden.

Move type 2: sequencing move

The sequencing move, performed by the job sequencing on the same families/batches as well as the batch sequencing on a machine(s), changes the job order, batch order, and machine assignment and, consequently, the batch composition might change. Therefore, the sequencing moves implemented by the ITS and OTS, determine, respectively, the best job sequence within batches as well as the best batch sequences on machines.

The neighborhoods for the OTS are generated by moving the batches, yet one move at a time. The benefits of these moves consist of re-sequencing newly formed batches after performing an insert/swap move at the upper level (the CTS). Another benefit lies in the fact that these moves are capable of changing the batch composition(s) by merging batches. Based upon swap-related move, the swap move changes the positions of two batches by maintaining the same positions for the remaining batches either on the same machine or different machines, regardless of which batches of the same group or different groups will be swapped. Based upon insertion-related move, the insert move removes a batch scheduled to be processed on a machine from its current position and inserts it in any available position, either on the same machine or different machines.

Basically, the neighborhood for the ITS are similarly defined as those for the OTS. However, the neighborhood mechanisms are restricted only to the jobs within each batch. Note that performing these types of moves only alter the job sequence in each batch and refine the schedule after applying the OTS. It is worth noting that for any combination between batch compositions, the GTAs are not violated by performing inside moves.

It is worth noting that during the generation of neighbor solutions in the central and outside search levels, the machine eligibilities and/or LB_{ih}^{k} must be considered for processing a moved job(s)/batch(es) as well as a newly formed batch(es). Apart from this, the neighborhood search mechanisms are the same for all search levels and performed at each level in order to determine *simultaneously* the best neighbor solutions at the same level as well as the upper level of the search. In other words, the ITS is performed on each OTS neighbor solution in order to obtain the best job sequence within each batch of each OTS neighbor solution so that the best OTS neighbor solution is identified. Likewise, the OTS is performed on each CTS neighbor solution in order to obtain the best batch sequence on each machine of each CTS neighbor solution so that the best CTS neighbor solution is identified.

4.2.4.5. Tabu List Structures

TS applies tabu list as the Short-Term Memory (STM) function in order not to be trapped in a local optima. The STM prohibits the search from choosing some of the previously identified neighborhoods to avoid being trapped in a local optima. Due to involving different types of operations related to neighborhood mechanisms at each level of the search algorithms, different types of tabu structures are implemented.

The tabu structure in the ITS is based upon positions of a job(s) related to a batch belonging to a group. In other words, if a job belonging to S_{si} is inserted from its current position p to another position p' $(p \neq p')$, the sequencing move $v_I(S_{si}|p',p)$ is stored in the ITL. $v_I(S_{si}|p',p)$ indicates that a job belonging to S_{si}

cannot be inserted from its current position p to a new position p' for several predetermined iterations. Apart from this, if a job belonging to S_{si} in position p is exchanged with another job belonging to the same batch of the same group in position p' $(p \neq p')$, the sequencing move $v_E(S_{si}|p', p)$ is stored in the ITL. Likewise, $v_E(S_{si}|p', p)$ indicates that a job belonging to S_{si} in position p cannot be swapped with another job belonging to the same batch in position p' for several predetermined iterations.

The tabu structure in the OTS is based upon positions of a batch(es) on a machine(s). In other words, if a batch assigned to machine m is inserted from its current position p to another position p' $(p \neq p')$ on machine m', the sequencing move $v_I(m'_{p'}, m_p)$ is stored in the OTL. $v_I(m'_{p'}, m_p)$ indicates that a batch assigned to position p on machine m cannot be inserted to position p' on machine m' for several predetermined iterations. Apart from this, if a batch assigned to machine m in position p is exchanged with another batch assigned to machine m' in position p' (if $m = m'$ then $p \neq p'$), the sequencing move $v_E(m'_{p'}, m_p)$ is stored in the OTL. Likewise, $v_E(m'_{p'}, m_p)$ indicates that a batch assigned to position p on machine m cannot be swapped with another batch assigned to position p' on machine m' for several predetermined iterations.

Finally, the tabu structure in the CTS is based upon triple groups involved in a move and is stored in the CTL as a triple group sequential on a machine. Triple group sequential is referred to three groups that are not necessarily different from each other, and are related to three jobs which are processed one after another on the same machine. In other words, if j_{sij} is inserted between two jobs belonging to groups i' and i'' on machine m (i.e. $j_{s'i'j'}$ and $j_{s"i"j"}$ are processed immediately before and after j_{sij}, respectively), then the dividing/sequencing move $v_I(m|g_{i'}, g_i, g_{i''})$ is stored in the CTL. $v_I(m|g_{i'}, g_i, g_{i''})$ indicates that a job belonging to group i cannot be inserted between two jobs, which belong to groups i' and i'' on machine m for several predetermined iterations. Apart from this, if j_{sij}, which is processed immediately before and after jobs of groups i' and i'' on machine m, respectively, is exchanged with another job of group p, which is processed immediately before and after jobs of groups p' and p'' on machine m', respectively, then the dividing/sequencing move $v_E(m_{g_{i'},g_p,g_{i''}}|m'_{g_{p'},g_i,g_{p''}})$ is stored in the CTL. Likewise, $v_E(m_{g_{i'},g_p,g_{i''}}|m'_{g_{p'},g_i,g_{p''}})$ indicates that a job of group i cannot be swapped with another job of group p, for several predetermined iterations, so that a job of group i is peocessed between two jobs of group p' and p'' on machine m' and a job of group p is processed between two jobs of groups i' and i'' on the same machine. If there is no group before the inserted job, the reference group is used as the first group in a triple group sequential. Likewise, if there is no group after the inserted job, the group assigned to the inserted job is used as the last group in a triple group sequential. In all tabu formulation above, v_I and v_E stand for insertion-related and swap-related operator, respectively. Figure 11 illustrates an example of tabu list as well as neighborhood structures by applying insertion-related move on the CTS followed by swap-related move on the OTS, which itself is followed by swap-related move on the ITS.

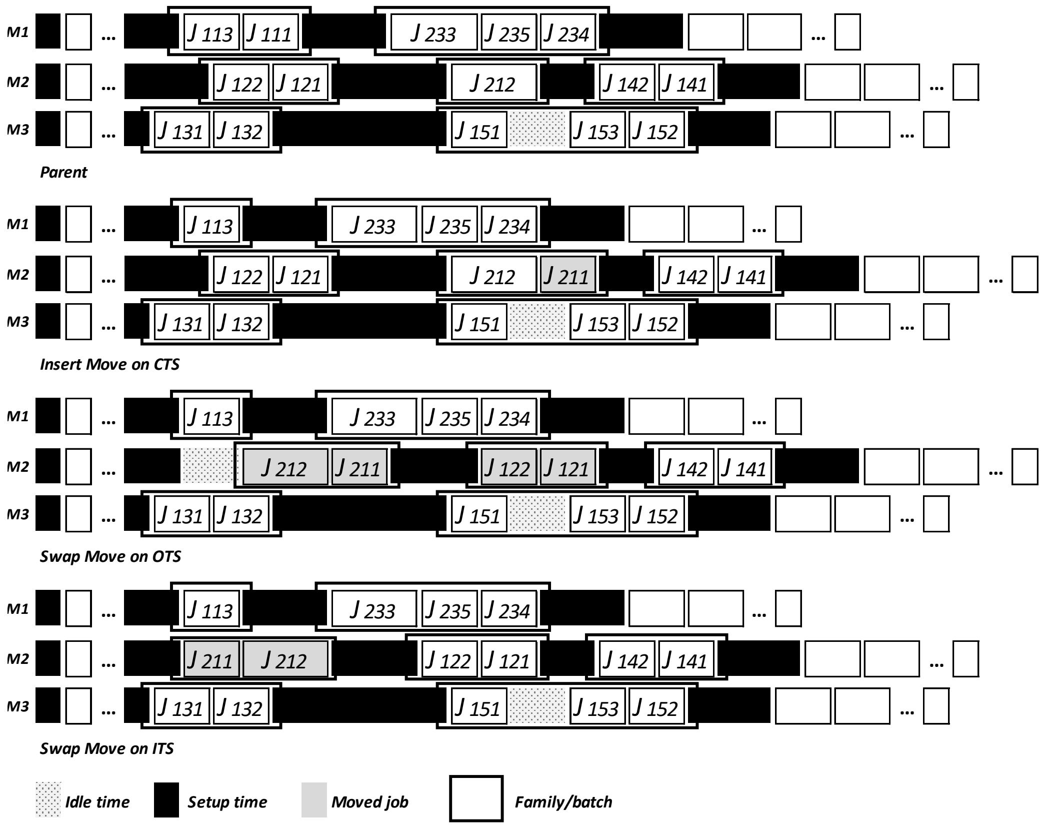


Figure 11. Illustration of neighborhood and tabu structures in different levels of TS-based algorithm

By inserting J_{111} to m_2 in the CTS and applying the refinement step, the completion times of J_{233}, J_{234}, and J_{235} are decreased, while the completion times of J_{211}, J_{141}, and J_{142} are increased. The move $v_I(2|1,1,4)$ is stored in the CTL. Also, by swapping S_{12} and S_{21} on m_2 in the OTS, the completion times of J_{211} and J_{212} are decreased, while the completion times of J_{121} and J_{122} are increased. It is worth noting that by applying this move, an idle time is created on m_2 due to the release time of J_{212}. In addition, setup times of the first batch of the second group and the second batch of the first group are changed. The move $v_E((m_2)_{2+k},|(m_2)_{1+k})$ is stored in the OTL. k represents the number of batches processed before J_{122} in the parent. Finally, the completion times of J_{211}, J_{121}, J_{122}, J_{141}, and J_{142} are decreased by exchanging positions of J_{211} and J_{212} on m_2 in the ITS. Then, the move $v_E(S_{21}|2,1)$ is stored in the ITL. In all levels, the completion times of jobs which are not mentioned are held fixed. Also, the tardiness of a job, due to its completion time being decreased, is either decreased or not changed. Likewise, the tardiness of a job, due to its completion time being increased, is either increased or not changed. As a result, the objective function value of the solution created by a combination of those moves is decreased, even though the completion time and/or tardiness of J_{121} and J_{122} are increased.

4.2.4.6. Algorithmic steps of three-level TS

Step 1 (IS): Determine a random feasible IS or an IS with the help of an IS finding mechanism as a combination of the CSD^k, OSD^k, and ISD^k, $\forall\ k \in m$.

Step 2 (CTS): Enter all neighbor solutions of the CSD^k generated with the help of insertion and/or swap operators into a list named central temporary candidate list (CTCL).

Step 3 (Refinement): Modifying the batch combination, the refinement step is needed for each neighbor solution after applying insertion and/or swap operators in the CTS. This step (that leads to the refined seed in the CSD, OSD and ISD parts) includes: modifying the number of batches belonging to each group together with the number and the type of jobs belonging to each batch (the batch combination), and modifying batch sequencing on machine(s) as well as job sequencing within each batch. In other words, the OSD and ISD sequences related to a machine(s), which is a candidate for a move in the CTS, are changed when the sequence in the CSD changes on the machine(s). Therefore, the refined seed for the CSD, OSD and ISD parts (known as the refined CSD, OSD, and ISD) is achieved after applying the insertion and/or swap operators and then the refinement step on the CSD in the CTS.

Step 4 (OTS): The OTS is implemented for the best / all / part of neighbor solution(s) in the CTCL in order to find the best sequence of batches associated with this solution(s). In each iteration of the OTS, the batch sequence and job sequence within batches of each refined CSD will be considered as the OSD and ISD, respectively. The following steps illustrate the OTS applied for each refined CSD.

Step 4.1: Enter all neighbor solution of the OSD generated with the help of insertion and swap operators into a list named outside temporary candidate list (OTCL).

Step 4.2 (Refinement): Similar to that explained for the refinement step in the CTS, the refinement step might be needed after applying insertion and swap operators in the OTS. This step includes: modifying batch sequence on each machine, job sequence on each batch, and the batch combination.

Step 4.3 (ITS): The ITS is implemented for the best / all / part of neighbor solution(s) in the OTCL in order to find the best sequence of jobs within batches associated with this solution(s). In each iteration of the ITS, the job sequence within batches of each refined OSD will be considered as the ISD. The following steps illustrate the ITS applied for each refined OSD.

Step 4.3.1: Enter all neighbor solutions of the ISD generated with the help of insertion and swap operators into a list named inside temporary candidate list (ITCL).

Step 4.3.2: Update the following parameters for the ITS. The job sequence of the refined OSD in the OTS (i.e., the refined ISD) is considered as the first member of the inside candidate list (ICL) and inside index list (IIL). The objective function value of this solution is set as the inside aspiration level (IAL). As it is mentioned, the IAL is the best value found among all the neighbor solutions of the current inside neighborhood search. Then, the first best solution in the ITCL is inserted into the ICL, only if the ICL has not included this solution and the inside tabu list (ITL) has not included the move, which led to this solution; otherwise the next best solution is inserted into the ICL. This remains true unless the forbidden move leads to a solution better than the IAL. Each solution in the ICL is called a candidate and all of these solutions should be distinct from each other. Also, the neighborhoods included in the candidate list must be excluded from further consideration. If more than one neighborhood has the same value, then the neighborhood with the first best solution is chosen. Tabu list (TL) is used to avoid the cyclical problem by forbidding certain moves which are called tabu. All moves in the TL will be considered as forbidden moves for a certain number of iterations, which is known as inside tabu list size (ITLS). The ITL is updated and the move which led to an entry into the ICL is stored in the ITL. So if the ITL is filled to its capacity, the move which was forbidden for the longest duration (the oldest move in the ITL) is replaced with the most recent move in the ITL. After each insertion into the ICL, the inserted candidate will be compared to the previous candidate. If the objective function value has improved then the current candidate will be assigned a star (*), which indicates it is capable of becoming the next local optimum. If the objective function value has not improved and the previous entry into the ICL already has a star, it will be assigned another star (**) and entered into the inside index list (IIL). The IIL is a subset of the ICL, which includes the local optima.

Step 4.3.3: When a feasible solution is added to the ICL, and its value is not less than the value of the previous member of the ICL, the value of inside iterations without improvement (IIWOI) is increased by one; otherwise, this counter is reset to zero. The ITS will stop, whenever the IIWOI attains the maximum inside iterations without improvement (MIIWOI), or a certain number of local optima, referred to as inside index list size (IILS), reaches to the maximum inside index list (MIIL). If either of these criteria attains its predefined values, the search will be stopped, and the best sequence of jobs within each batch is considered as the best solution for the current batch sequence and the search is switched to the OTS; otherwise, the last entry in the ICL will be considered as the next seed for the ITS and the search will be directed to step 4.3.1.

Step 4.4 (back to the OTS): The batch sequence as well as the job sequence belonging to each batch of the refined CSD in the CTS (i.e., the refined OSD and ISD) is considered as the first member of the outside candidate list (OCL) and outside index list (OIL). It is necessary to hold both the OSD and ISD as one member of the OCL and OIL, because the ISD might be changed when the OSD changes. Similar to the ITS in step 4.3.2, the OTS is performed and the outside aspiration level (OAL), the OCL, the outside tabu list (OTL), and the OIL are updated.

Step 4.5: The OTS will stop, whenever the outside iteration without improvement (OIWOI) attains the maximum outside iteration without improvement (MOIWOI), or a certain number of local optima, referred to as outside index list size (OILS), reaches to the maximum outside index list (MOIL). If either of these criteria attains its predefined values, the search will be stopped, and the best sequence of batches as well as the best sequence of jobs within batches (determined in the ITS) are considered as the best solution for the current batch combination and the search is switched to the CTS; otherwise, the last entry into the OCL will be considered as the next seed for the OTS and the search will be directed to step 4.3.

Step 5 (back to the CTS): The IS composed of the CSD, OSD, and ISD is considered as the first member of the central candidate list (CCL) and the central index list (CIL). It is necessary to hold the CSD, OSD, and ISD as one member of the CCL and CIL, because the OSD and ISD are changed when the CSD changes. Similar to the ITS in step 4.3.2, the CTS is performed and the central aspiration level (CAL), the CCL, the central tabu list (CTL), and the CIL are updated.

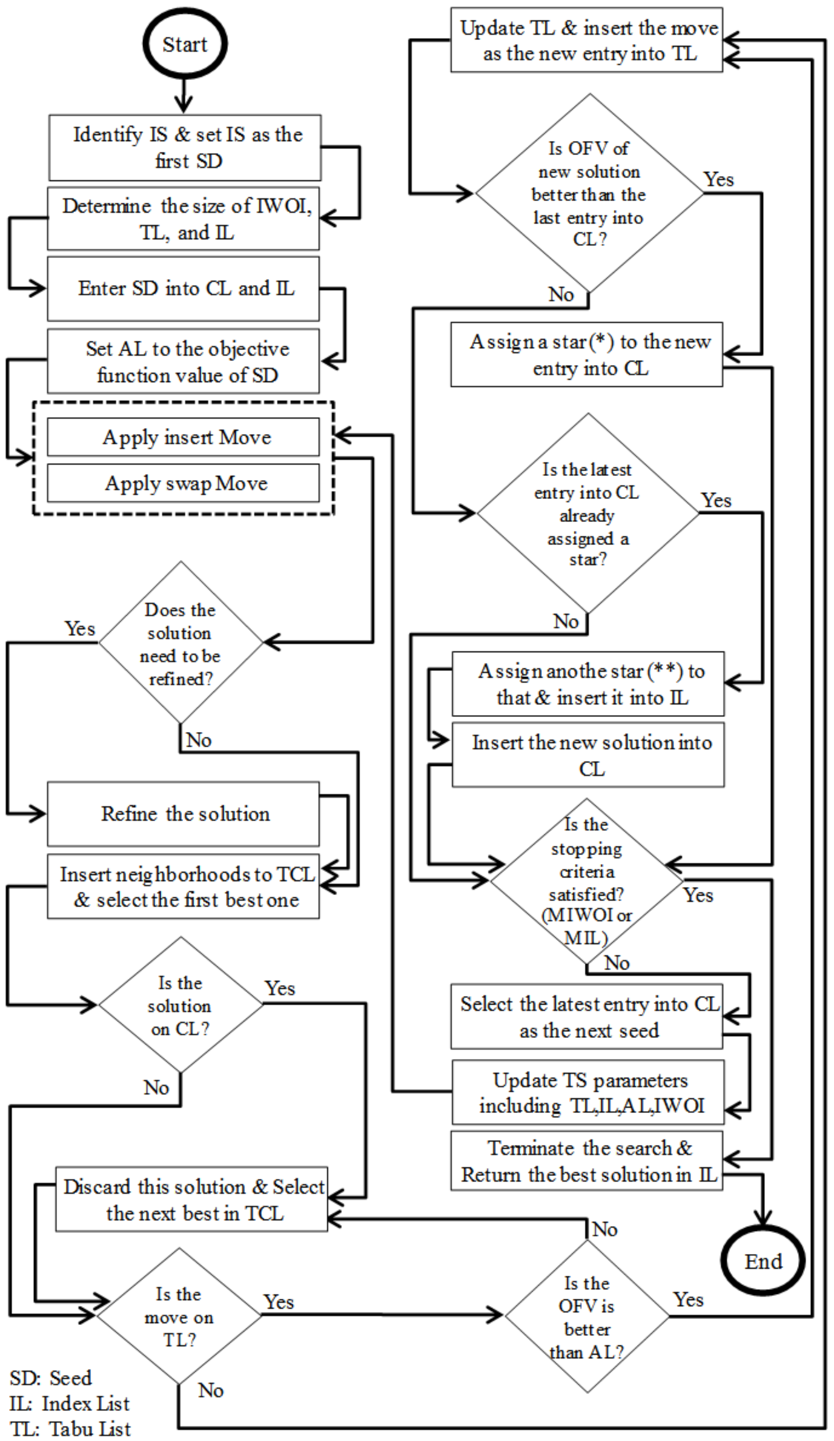
Step 6: The CTS will stop, whenever the central iteration without improvement (CIWOI) attains the maximum central iteration without improvement (MCIWOI), or a certain number of local optima, referred to as central index list size (CILS), reaches to the maximum central index list (MCIL). If either of these criteria attains its predefined values, the search will be stopped, and the search algorithm is switched to the next stage of the LTM-MAX until it attains the maximum frequency of central search level; otherwise, the last entry into the CCL will be considered as the next seed for the CTS and the search will be directed to step 2.

It is worth mentioning that the algorithmic steps of TS, from step 1 to 6, should be consecutively applied for each stage of HFS, when the stage-based interdependency strategy is implemented. The pseudo-code for the proposed TS-based algorithm with three search levels as well as the flowchart for each level of the search algorithm (the CTS, OTS, and ITS) are depicted in Table 3 and Figure 12, respectively.

Table 3. Pseudo-code for three-level TS-based algorithm

TS Algorithm: Outline of TS-based algorithm

```
1:  Input: $S^{Seed}$
2:  Output: $The\ best\ objective\ function\ value\ \&$
            $The\ best\ schedule\ S^{*}\ related\ to\ a\ seed$
3:  $Decompose\ S^{Seed}\ as\ CSD, OSD, and\ ISD$
4:  while $(CILS \le MCIL\ ||NIWOI \le MIWOI)$ do
5:  $Perform\ Central\_Search()$
6:    $Update\ CTS\ parameters: CTL, CIL, NIWOI, and\ CAL$
7:    $TCCL \leftarrow Generate\_neighbor\_solution(S^{Seed})$
8:    for $S^{k} \in TCCL, k = \{1, \dots, n_{TCCL}\}$ do
9:      $TCCL \leftarrow Refine\ Solution()$
10:   end for
11:   $Go\ to\ the\ Outside\ Search\ for\ each\ central$
      $neighbor\ solution$
        // The same algorithmic steps followed by CTS
12:     $Perform\ Outside\_Search()$
13:     $Go\ to\ the\ Inside\ Search\ for\ each\ outside$
        $neighbor\ solution$
          // The same algorithmic steps followed by CTS
14:       $Perform\ Inside\_Search()$
15:   $S^{best} \leftarrow Identify\ the\ best\ entry\ in\ TCCL$
      // Check $S^{best}$ vs CTL & CCL with respect CAL
16:   if $((move_{S^{best}} \notin move_{Tabu}\ \&\&\ S^{best} \notin CCL)\ ||$
         $(S^{best} \notin CCL\ \&\&\ f(S^{best}) < CAL))$ then
17:     $CCL \leftarrow S^{best}$
18:     $CTL \leftarrow move_{S^{best}}$
19:     $TCCL = \emptyset$
20:   else
21:     $S^{best} \leftarrow Find\ next\ best\ entry\ into\ TCCL$
22:     $Go\ to$ if $condition$
23:   end if
      // $S^{PE_{CCL}}$: The entry into CCL immediately before $S^{best}$
24:   if $(S^{best} \le S^{PE_{CCL}})$ then
25:     $S^{best^{*}} \leftarrow S^{best}$
26:     $S^{Seed} \leftarrow S^{best}$
27:   else if $(S^{PE_{CCL}}\ has\ a\ star\ (S^{PE_{CCL}^{*}})\ \&\&$
            $f(S^{best}) > f(S^{PE_{CCL}}))$ then
28:     $S^{PE_{CCL}^{**}} \leftarrow S^{PE_{CCL}^{*}}$
29:     $CIL \leftarrow S^{PE_{CCL}^{*}}$
30:     $S^{Seed} \leftarrow S^{best}$
31:   Else
32:     $S^{Seed} \leftarrow S^{best}$
33:   end if
34: end while
35: return $S^{*}$
```


Start
Identify IS & set IS as the first SD
Determine the size of IWOI, TL, and IL
Enter SD into CL and IL
Set AL to the objective function value of SD
Apply insert Move
Apply swap Move
Does the solution need to be refined?
Yes
No
Refine the solution
Insert neighborhoods to TCL & select the first best one
Is the solution on CL?
Yes
No
Discard this solution & Select the next best in TCL
Is the move on TL?
Yes
No
Is the OFV is better than AL?
Yes
No
Update TL & insert the move as the new entry into TL
Is OFV of new solution better than the last entry into CL?
Yes
No
Assign a star (*) to the new entry into CL
Is the latest entry into CL already assigned a star?
Yes
No
Assign anothe star (**) to that & insert it into IL
Insert the new solution into CL
Is the stopping criteria satisfied? (MIWOI or MIL)
Yes
No
Select the latest entry into CL as the next seed
Update TS parameters including TL,IL,AL,IWOI
Terminate the search & Return the best solution in IL
End

SD: Seed
IL: Index List
TL: Tabu List
IS: Initial Solution
CL: Candidate List
MIL: Maximum Index List
AL: Aspiration Level
TCL: Temporary Candidate List
IWOI: Iteration Without Improvement
MIWOI: Maximum Iteration without Improvement
OFV: Objective Function Value

Figure 12. Flow chart for basic TS at each level

4.2.4.7. The level moving in three-level TS

In dealing with the batch scheduling problem presented in this research, the TS-based algorithms have been employed in three levels: the CTS, OTS, and ITS. There are three different ways to move between these levels. The first method (M1) focuses on all possible neighbor solutions by performing the ITS and OTS for any individual neighbor solution generated in the OTS and CTS, respectively, until no more improvement is achievable in the CTS, whereas the second method (M2) performs the ITS and OTS for Q% and K% of all possible neighbor solutions generated in the OTS and CTS, respectively. The Q% and K% represent the solutions with the best (smallest) objective function value amongst all solutions (promising solutions), before performing the OTS for any individual neighbor solution generated in the CTS and performing the ITS for any individual neighbor solution generated in the OTS, respectively. The Q% and K% guarantees to obtain the same best solution as the first method at a 5% significance level, but in drastically shorter computational time. The third method (M3) passes the best neighbor solution obtained by the ITS to the OTS and also the OTS to the CTS and vice versa, until no more improvement is achievable at any level of TS. The iterative levels includes: finding the best job order for a given batch composition and batch order in the ITS; finding the best batch order for a given batch composition and job order in the OTS; and finally finding the best batch composition for a given batch order and job order in the CTS.

Based upon randomly generated test problems in each problem size and with respect to the minimum deviation between the first and second methods in the level moving, the average Q% and K% determined are presented in Table 4. At a 5% significance level, a one-way hypothesis test is used for each problem size to investigate whether or not the average percentage is smaller than the determined value. Based upon a large P_{value} corresponding to each hypothesis in Table 4, the results indicate the second algorithm performs the best at the mentioned Q% and K%. Also, 95% Confidence intervals (CI) are determined for each problem size in Table 4.

Table 4. Percentage values and confidence intervals for Q and K

Problem size		Value %	95% CI	P_{value}
Small-size	*Q%* :	25%	[22.45 ,27.54]	1.464E-08
	K% :	20%	[17.06, 22.94]	1.727E-08
Medium-size	*Q%* :	32%	[28.48, 35.52]	1.318E-08
	K% :	26%	[21.88, 30.12]	1.345E-08
Large-size	*Q%* :	38%	[35.06, 40.94]	1.973E-08
	K% :	30%	[24.12, 35.88]	1.273E-08

4.2.5. Tabu search/Path-Relinking

The lack of mechanism for exploring the information on good quality solutions becomes more pronounced when batch scheduling problems in HFS are accompanied by the stage-based interdependency strategy.

These arguments led us to probe into the properties of a population-based structure and, consequently, develop a TS-based algorithm enhanced with a population-based structure. Tabu search/path-relinking (TS/PR) repeatedly operates back and forth between path-relinking (PR) and TS for a pre-determined number of iterations without any improvement.

PR was originally proposed by Glover (1997) as an approach to integrate diversification and intensification strategies in the search (LTM-MAX & LTM-MIN). The PR procedure is implemented to explore trajectories connecting elite solutions in both directions, which are obtained by tabu search or scatter search (Glover et al. 2000). The solution that begins the path is called an *initial solution* (S^I), while the solution that the path leads to is called a *guiding solution* (S^G). PR consists of gradual introduction of attributes of S^G in S^I. The *InitialPathSet* is a list of all intermediate solutions generated during PR, while the *PromisingPathSet* is a list of candidate solutions, which are a subset of the *InitialPathSet*. After the relinking procedure, a so-called reference solution is chosen from the *PromisingPathSet* that serves to update the population. TS/PR stops after a predetermined number of iterations without any improvement. PR mainly integrates two complementary key components to ensure search efficiency:

- the construction approach used for establishing the path between S^G and S^I; and
- the method used to choose the reference solution (Peng et al. 2015).

Therefore, PR is incorporated into the basic TS to increase its performance. In other words, the PR procedure explores trajectories connecting elite solutions between S^G in S^I in a relinking path, in both directions ($S^I \rightleftarrows S^G$), while TS improves the generated promising solution to a local optimum (Peng et al. 2015). The following sections present the algorithmic steps as well as the characteristics of the PR in detail, with respect to the stage-based interdependency strategy.

4.2.5.1. Solution representation for PR

In PR, a partial solution is represented by v^k permutation arrays in k^{th} stage, so that each permutation array represents an ordering of operations on h^{th} machine in k^{th} stage ($\forall\, h \in v^k$). A partial solution of HFS in k^{th} stage of PR is represented as follows:

$$S^k = \left\{\left(j_{1,1}^{s^k}, j_{2,1}^{s^k}, \dots, j_{\mathcal{J}_1,1}^{s^k}\right), \left(j_{1,2}^{s^k}, j_{2,2}^{s^k}, \dots, j_{\mathcal{J}_2,2}^{s^k}\right), \dots, \left(j_{1,v^k}^{s^k}, j_{2,v^k}^{s^k}, \dots, j_{\mathcal{J}_h,v^k}^{s^k}\right)\right\}$$

$\mathcal{J}_h$ represents the number of operations processed by machine h, i.e., $p \in \{1,2,\dots,\mathcal{J}_h\}$, $\forall\, h \in v^k$, i.e., $\sum_{h=1}^{v^k} \mathcal{J}_h = N^k$. $j_{p,h}^{s^k}$ represents the p^{th} operation processed by machine h related to a partial solution in k^{th} stage (S^k), where an operation ($j_{p,h}^{s^k}$) indicates a job of a group ($j_{p,h}^{s^k} = ij$, $i \in G$ & $j \in G_i$). The S^k is similar to the CSD^k, regardless of which batch of a group a job is assigned to. Figure 13 shows an example sequence presented in section 4.2.4.3. The first operation processed by machine 1 is related to the third job of group 1 ($j_{1,1}^{s^k} = 13$), the first operation processed by machine 3 is related to the first job of group 5 ($j_{1,3}^{s^k} = 51$), and so on. We will use this solution representation to develop path construction.

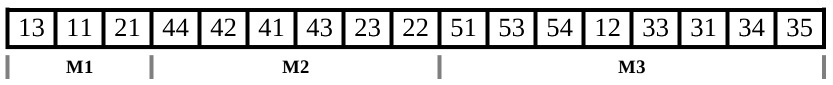


Figure 13. Solution representation for PR

4.2.5.2. Initial population

In each stage, the initial population (IP) is constructed in order to trigger TS/PR into the solution space as follows:

I. Construct the IS based on the IS finding mechanism developed in section 4.2.2.
II. Optimize the IS to become a local optima with the help of strong TS.
III. Select $P_{size} - 1$ solutions randomly from the IL and CL of the CTS (CIL & CCL) and add them along with the improved IS (best sequence) to the IP.

P_{size} represents the size of the IP, which is equal to 10 in this research. If the CIL size (CIL_{SIZE}) is less than $P_{size} - 1$, the other members of the IP ($P_{size} - CIL_{SIZE} - 1$) are selected from the central candidate list (CCL). At each iteration of TS/PR, the two improved solutions obtained by implementing TS/PR on a pair of S^I and S^G in both directions ($S^I \rightleftarrows S^G$), are replaced by the two worst solutions in the IP, if it does not duplicate any solution currently in the IP. The pseudo-code for the IP is presented in Table 5.

Table 5. Pseudo-code for the IP generation

```
$S^{IS} \leftarrow Identify\ Initial.Solution()$ //Section 4.2.2
$S^{IMP} \leftarrow Central.Tabu.Search(S^{IS})$ //Section 4.2.4.2
$IP \leftarrow S^{IMP}$
$IP \leftarrow Select\ (P_{size} - 1)\ random\ solutions\ from\ the\ CIL$
**if** $P_{size} > (CIL_{SIZE} + 1)$ **then**
  $IP \leftarrow Select\ (P_{size} - CIL_{SIZE} - 1)\ NonRepeated\ random\ solutions\ from\ the\ CCL$
  **if** $P_{size} > (CCL_{SIZE} + 1)$ **then**
    **for each** $i \in \{1,2,\dots,(P_{size} - CCL_{SIZE} - 1)\}$ **do**
      $Generate\ S^i\ randomly$
      $IP \leftarrow Adjustment.Step(S^i)$ //Section 4.2.3
    **end for**
  **end if**
**end if**
// CIL: central index list; CCL: central candidate list
```

4.2.5.3. Path construction

After randomly choosing S^I and S^G from the *PairSet*, which includes pairs of elite solutions of the IP and then checking a pair solutions against *TabuSet*, we can generate a path to link aforementioned solutions. *TabuSet* records all pairs of solutions selected during the search procedure and prevents PR to select them again. According to the neighborhood operators including swap and insert moves, distance or dissimilarity measure, the position of the candidate jobs and their corresponding insertion points, a path is constructed. Among many types of moves considered in the literature for the flow shop problems, the swap and the insertion operators appear prominently (Nowicki and Smutnicki 1996). Taillard (1993) showed that the insertion operator is more effective than the swap operator when used in a neighborhood search. Therefore, three path constructions are developed in terms of insertion- and swap-related operators: longest common subsequence-based and block-based constructions based on insertion-related operator along with swap-based construction based on swap-related operator. In the following, the aforementioned path constructions are explained.

LCS-based construction: In this method, a longest common subsequence (LCS) is chosen as a distance measure to construct a path between S^I and S^G, for a stage with multiple machines. The distance in the LCS indicates the minimal number of moves required to link S^I and S^G (Basseur et al. 2005, Zeng et al. 2013). Correspondingly, the distance (d) depends on the length of the LCS (l) shared by S^I and S^G in k^{th} satge, in which both l and d vary in the interval $[1, N^k]$ and $(d + l) \geq \sum_{i \in I^k} n_i^k$. The LCS can be calculated in $O(N^2)$ by a dynamic programming algorithm, which is similar to the well-known Needleman-Wunsch algorithm (Cormen et al. 1990, Schiavinotto and Stützle 2007). The LCS corresponding to S^I and S^G is computed by the following iterative procedure:

- **Itr 1:** Obtain the smallest value of $p + q$ when $j_{p,h}^{(s^I)^k} = j_{q,l}^{(s^G)^k}$, regardless of job assignments on machines (h and l). A tie is broken in favor of S^I;
- **Itr 2:** Determine the forward minimum distance between $j_{p,h}^{(s^I)^k}$ & $j_{p+1,h'}^{(s^I)^k}$ in S^G and $j_{q,l}^{(s^G)^k}$ & $j_{q+1,l'}^{(s^G)^k}$ in S^I, regardless of job assignments on machines (h, h', l and l');
- **Itr 3:** Select the jobs corresponding to the initial and final positions on the forward minimum distance, determined in the second iteration, in both S^I and S^G, as the jobs belonging to the LCS;
- **Itr 4:** Replace p & q by the last selected positions of the LCS in S^I and S^G and go to the first iteration, until $p = \sum_{i \in I^k} n_i^k$ and/or $q = \sum_{i \in I^k} n_i^k$.

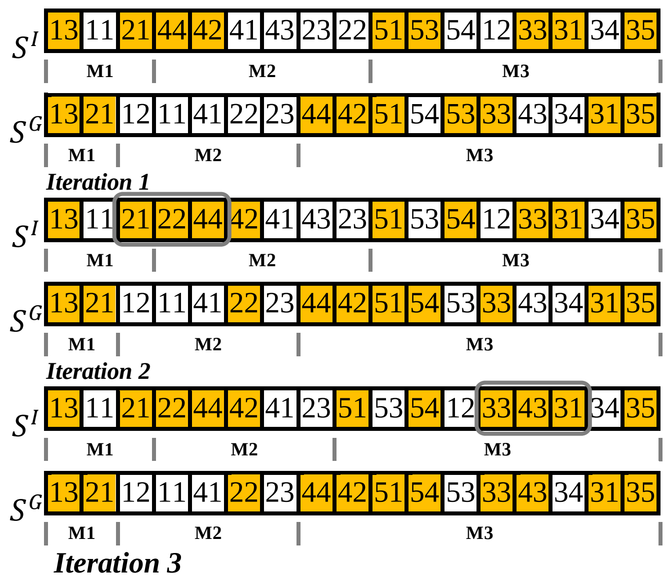
Since the LCS is not unique in most cases, only one of them is randomly chosen to generate the path. Besides, the jobs, which do not belong to the LCS, are called the candidate jobs. If $[j]_{x,y}^{s^k}$ represents candidate job j assigned to position x of machine y in S^k, the LCS-based construction develops solutions belonging to a relinking path of S^I to S^G by the following steps:

- **Step 0:** Compute the LCS between S^I and S^G;
- **Step 1:** Determine all possible insertion points for all the candidate jobs as follows:
  - If $j_{q,l}^{(s^G)^k} \leq [j]_{x,y}^{(s^G)^k} \leq j_{q',l'}^{(s^G)^k}$, when $x \in [q, q']$, $y \in [l, l']$, and $j_{q,l}^{(s^G)^k}$ & $j_{q',l'}^{(s^G)^k}$ are two *neighboring jobs* in the LCS, insert the candidate job so that $j_{p,h}^{(s^I)^k} \leq [j]_{x,y}^{(s^G)^k} \leq j_{p',h'}^{(s^I)^k}$, when $x \in [p, p']$, $y \in [h, h']$, $j_{q,l}^{(s^G)^k} = j_{p,h}^{(s^I)^k}$, $j_{q',l'}^{(s^G)^k} = j_{p',h'}^{(s^I)^k}$, and $j_{p,h}^{(s^I)^k}$ & $j_{p',h'}^{(s^I)^k}$ are two *neighboring jobs* in the LCS.
  - If $j_{p,h}^{(s^I)^k} \leq [j]_{x,y}^{(s^G)^k} \leq j_{(p+1),(h+1)}^{(s^I)^k}$ $\forall h \in (V^k - \{1\})$, two insertion points are determined depending on different job assignments to machines.
  - Consider the position located before the first job/after the last job as the insertion point for the job in the beginning/at the end of the permutation.
- **Step 2:** Analyze all feasible candidate moves incorporating attributes of S^G with respect to all possible insertion points determined in the first step;
- **Step 3:** Choose one feasible candidate move as a current solution (S^C) by random selection from global and/or local optima related to feasible candidate moves determined in the second step;
- **Step 4:** Enter S^C into the *InitialPathSet*, $S^I \leftarrow S^C$, and $d \leftarrow (d-1)$; and
- **Step 5:** Go to step 0, until $d = 0$.

Although there are usually several possible insertion points for all candidate jobs (at least one insertion point for each), only one of them is selected for path generation in each step. Therefore, in each step of the LCS-based construction, the LCS is computed, then all feasible candidate moves incorporating attributes of S^G are analyzed with respect to all possible insertion points, and finally one feasible candidate move is chosen by random selection from global and/or local optima related to feasible candidate moves. The first insertion, last insertion, and random insertion are other methods to choose a move. The distance between S^C and S^G is decreased by 1, after inserting the selected move in each step. Since the comparison is done before generating a new intermediate solution S^C, the time complexity of the method is $O(N^3)$ (Zeng et al. 2013).

There is at least d new solutions in a relinking path corresponding to a minimal path between S^I and S^G. Since a selected insertion point(s) must not violate the machine eligibilities and LB_{ih}^{k}, it increases the possibility to create a good path. As a result, S^C and S^I ($S^C \leftarrow S^I$) are different from each other in terms of the batch composition and/or machine assignment and/or batch order and/or job order.

An example of LCS-based construction related to the first two iterations is shown in Figure 14. First the distance between S^I and S^G is obtained by computing the LCS. In this example, the forward distance between $j_{1,1}^{(s^I)^k} = 13$ & $j_{2,1}^{(s^I)^k} = 11$ in S^G is 2 (jobs 21 and 12). The forward distance between $j_{1,1}^{(s^G)^k} = 13$ & $j_{2,1}^{(s^G)^k} = 21$ in S^I is 1 (only job 11). Therefore, the forward minimum distance is equal to 1 and, subsequently jobs 13 and 21 are considered as jobs belonging to the LCS, both in S^I and S^G. In the next iteration, by considering $p = 3$ and $q = 2$, the forward minimum distance is 0, related to $j_{3,1}^{(s^I)^k} = 21$ & $j_{1,2}^{(s^I)^k} = 44$, and job 44 is added to the LCS. After several iterations, the length of LCS, colored in yellow, calculates as $l = 9$, which includes job 13, 21, 44, 42, 51, 53, 33, 31, and 35. The remaining jobs are candidate jobs, including job 11, 12, 22, 23, 34, 41, 43, and 54. These jobs will be moved from their initial position in S^I in order to reach S^G, one move at a time.



S^I 13 11 21 44 42 41 43 23 22 51 53 54 12 33 31 34 35
M1 M2 M3
S^G 13 21 12 11 41 22 23 44 42 51 54 53 33 43 34 31 35
M1 M2 M3
Iteration 1
S^I 13 11 21 22 44 42 41 43 23 51 53 54 12 33 31 34 35
M1 M2 M3
S^G 13 21 12 11 41 22 23 44 42 51 54 53 33 43 34 31 35
M1 M2 M3
Iteration 2
S^I 13 11 21 22 44 42 41 23 51 53 54 12 33 43 31 34 35
M1 M2 M3
S^G 13 21 12 11 41 22 23 44 42 51 54 53 33 43 34 31 35
M1 M2 M3
Iteration 3

Figure 14. The LCS of initial and guiding solutions for the first three iterations

The possible insertion points of candidate jobs as well as the objective function related to each move in the first iteration are summarized in Table 6. There is two insertion points for job 11 due to different job assignments to machines. The new solution is obtained by removing job 22 from its current position

($j_{6,2}^{(s^I)^k}$ = 22), and then inserting it amongst jobs 21 and 44 in the first position processed by machine 2 ($j_{1,2}^{(s^C)^k}$ = 22). This solution, considered as the first entry into the *InitialPathSet*, was selected randomly between three global and local optima with the objective function values of 6814, 6982 and 7370, in the first iteration.

Table 6. Possible moves related to the first iteration

	The candidate job in S^I								
	12	11	41	22	23	54	43	34	
Insertion point	(21,44)	(21,44)	(21,44)	(21,44)	(21,44)	(21,44)	(51,53)	(33,31)	(33,31)
$f(move)$	6814	6953	7093	7318	6982	7670	7960	7370	7891

The length of LCS is increased by 1 ($l = 10$). In the next iteration, S^C will be replaced by S^I and the process will continue until the distance between S^I and S^G becomes zero. In the second iteration, by removing job 43 from machine 2 ($j_{5,2}^{(s^I)^k}$ = 43) and inserting it in machine 3 ($j_{6,3}^{(s^C)^k}$ = 43), the batch composition, machine assignment, batch order, and job order change. This solution is the second entry into the *InitialPathSet*.

Block-based construction: A block insertion-related operator is implemented for stages with single machine to construct a relinking path of S^I to S^G (Luo and Hu 2013) by the following steps:

- **Step 0:** Identify the minimum position p (in S^I) where $j_{p,1}^{(S^I)^k} \neq j_{p,1}^{(S^G)^k}$, by checking two jobs in the same position of S^I and S^G;
- **Step 1:** Identify a job block from position p (in S^I), which consists of the same consecutive jobs after job $j_{p,1}^{(S^I)^k}$ in S^G;
- **Step 2:** Insert the identified job block (in S^I) after the job that $j_{p,1}^{(S^I)^k}$ follows in S^G to identify S^C;
- **Step 3:** Enter S^C into the *InitialPathSet*, $S^I \leftarrow S^C$, and apply the adjustment step for S^C if it should be applied; and
- **Step 4:** Go to step 0, until $j_{p,1}^{(S^I)^k} = j_{p,1}^{(S^G)^k}$, $\forall\, p \in \{1, \dots, \sum_{i \in I^k} n_i^k\}$.

Figure 15 shows an example for this method. There is only one machine in k^{th} stage. First, $p = 2, j_{2,1}^{(S^I)^k} =$ 21, and the job block [21,12,11,41] is inserted after job 33 in S^I, which is followed by job 21 in S^G; second, $p = 2, j_{2,1}^{(S^I)^k}$ = 22, and the job block [22,23,44,42,51,54,53] is inserted after job 35 in S^I, which is followed by job 22 in S^G; and finally, $p = 7, j_{7,1}^{(S^I)^k}$ = 43, and the job block [43,34,31] is inserted after job 53 in S^I, which is followed by job 43 in S^G. Therefore, after three iterations, a path including two intermediate solutions is developed between S^I and S^G.

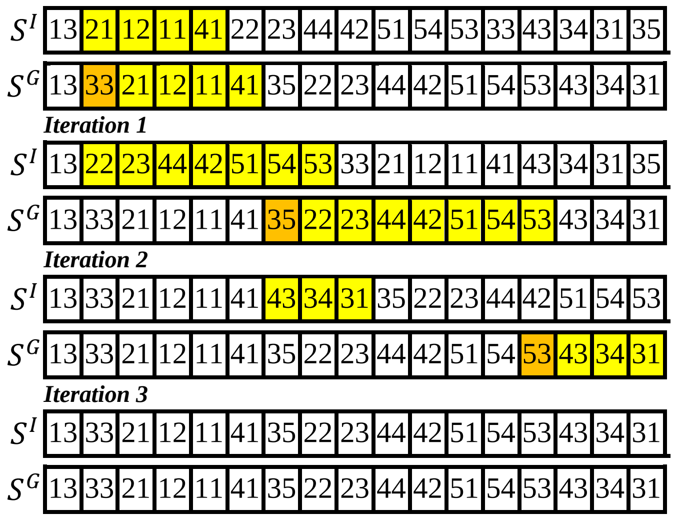

S^I 13 21 12 11 41 22 23 44 42 51 54 53 33 43 34 31 35
S^G 13 33 21 12 11 41 35 22 23 44 42 51 54 53 43 34 31
Iteration 1
S^I 13 22 23 44 42 51 54 53 33 21 12 11 41 43 34 31 35
S^G 13 33 21 12 11 41 35 22 23 44 42 51 54 53 43 34 31
Iteration 2
S^I 13 33 21 12 11 41 43 34 31 35 22 23 44 42 51 54 53
S^G 13 33 21 12 11 41 35 22 23 44 42 51 54 53 43 34 31
Iteration 3
S^I 13 33 21 12 11 41 35 22 23 44 42 51 54 53 43 34 31
S^G 13 33 21 12 11 41 35 22 23 44 42 51 54 53 43 34 31

Figure 15. Block-path construction method

This technique presents a good performance, particularly when $LB_{ih}^k \to n_i$. The drawback of this construction technique is to generate more infeasible S^C because of the block insertion-related operator during the path construction procedure, which may violate the machine eligibilities and/or LB_{ih}^k. This being the case, the infeasible solutions should be either abandoned or adjusted with the help of the adjustment step. None of these techniques are not a good approach to deal with infeasible solutions developed by this technique. Therefore, the following swap-based construction technique is developed to reduce the number of infeasible solutions generated in a relinking path.

Swap-based construction: A swap-related operator is implemented for stages with single machine to construct a relinking path of S^I to S^G (Peng et al. 2015). In each iteration, S^I is iteratively changed by exchanging two random operations that are in a different order in S^I and S^G, with the aim of reducing the number of different orders of operations in aforementioned solutions. Swap-based construction is implemented by the following steps:

- **Step 0:** Define the set of symmetric differences between S^I and S^G ($\delta^k_{S^I,S^G}$) based on job positions on the machine, i.e., $\delta^k_{S^I,S^G} = \left\{p \in \left\{1, \dots, \sum_{i\in I^k} n_i^k \right\} \mid j_{p,1}^{(S^I)^k} \neq j_{p,1}^{(S^G)^k}\right\}$;
- **Step 1:** Select randomly position p from set $\delta^k_{S^I,S^G}$, and swap $j_{p,1}^{(S^I)^k}$ with $j_{p',1}^{(S^I)^k}$ on S^I to generate S^C, where $j_{p',1}^{(S^I)^k} = j_{p,1}^{(S^G)^k}$, $\forall p' \in \left\{\left\{1, \dots, \sum_{i\in I^k} n_i^k \right\} - p\right\}$;

- **Step 2:** Update set $\delta^k_{S^I,S^G}$ by removing position p, enter S^C into the *InitialPathSet*, and $S^I \leftarrow S^C$, and apply the adjustment step for S^C, if it should be applied; and,
- **Step 3:** Go to step 0, until $\delta^k_{S^I,S^G} = \emptyset$.

The possibility of constructing a diversified path is increased because a random position in the set $\delta^k_{S^I,S^G}$ is chosen in each iteration of swap-related operator. Since there might be more violations on the machine eligibilities and/or LB^k_{ih}, the block- and swap-based constructions might not be able to construct a good relinking path of S^I and S^G for stages with multiple machines, even when the adjustment step is implemented. Instead, the LCS-based construction is a good approach to construct a path in stages with multiple machines. Therefore, TS/PR algorithms implement the block- or swap-based constructions for stages with single machine, which are accompanied by the LCS-based construction for stages with multiple machines.

4.2.5.4. Path solution selection

Each solution in a relinking path must be checked against the machine eligibilities and/or LB^k_{ih}. If a solution does not meet a machine eligibility(s) and/or a desired lower bound(s), the job assignment of this solution must be re-ordered based on the refinement step and/or adjustment step (Section 4.2.3). Each two consecutive solutions located at a relinking path differ only by inserting a job on a machine (Figure 16). Therefore, it is not productive to apply a time-consuming improvement procedure for all solutions in the *InitialPathSet*, since many of those solutions would lead to the same local optima.

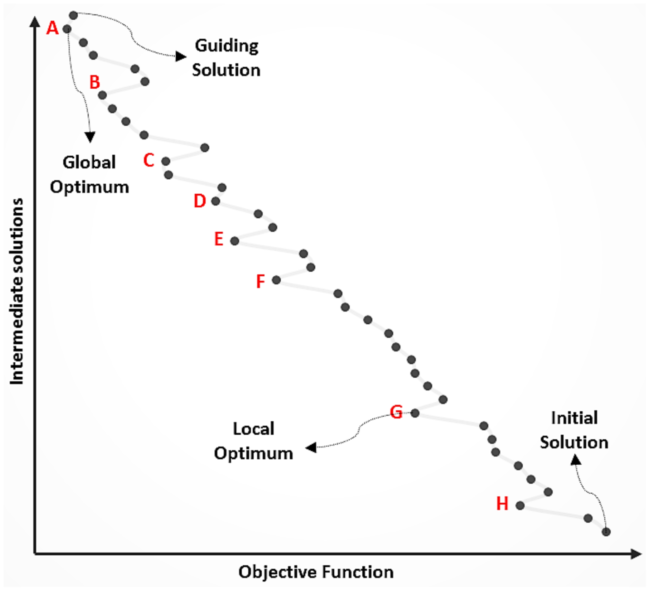


Figure 16. Global and local optima in intermediate solutions

Therefore, after the generation of all the intermediate solutions in the path, we select a set of global and/or local optima from the *InitialPathSet*, known as the *PromisingPathSet*, which is used to initialize a reference solution (S^R). As shown in Figure 16, the solutions *A*, *B*, *C*, *D*, *E*, F, and G are selected, since *A* is the global optimum and the others are local optimum.

4.2.5.5. Reference solution determination

The pseudo-codes for the PR procedure is depicted in Table 7. In order to avoid the problem of proximity of the local optima to S^I and S^G, a set of *K*-middle solutions of the *PromisingPathSet* is selected (Zeng et al. 2013).

Table 7. Pseudo-code for PR procedure

```
    PR Algorithm: Outline of the path-relinking procedure
1:  Input: $Initiating\ solution\ S^I, Guiding\ solution\ S^G, and\ v^k$
2:  Output: $A\ reference\ solution\ S^R$
    // Changes $S^I$ to $S^G$ by insertion operators: Section 4.2.4.4
3:  if $v^k = 1$ then //$v^k$ stands for # of machines in $k^{th}$ stage
      // Lines 4-11: Apply block-based construction
4:    $S^C = S^I, InitialPathSet = \emptyset$
5:    for $p = \{1, \dots, \mathcal{P}\}$ do
6:        if $j_{p,1}^{(S^I)^k} = j_{p,1}^{(S^G)^k}$ then
7:          $Countinue\ for$
8:        else
9:          $S^C \leftarrow Apply\ block-based\ insertion(S^I)$
10:         $InitialPathSet \leftarrow InitialPathSet \cup S^C$
11:   end for
12: else
      // Lines 13-22: Apply LCS-based construction
13:   $Identify\ the\ LCS\ and\ d$
14:   $S^C = S^I, InitialPathSet = \emptyset$
15:   for $l = \{d, \dots, 1\}$ do
16:       $Determine\ the\ insertion\ point(s)\ of\ a\ candidate\ job(s)\ in\ LCS$
17:       $Define\ the\ quality\ of\ each\ solution$
18:       $Local\ selection(a\ candidate\ job(s))$
19:       $S^C \leftarrow Randomly\ select\ an\ elemet\ of\ non-dominated\ solutions$
20:       $InitialPathSet \leftarrow InitialPathSet \cup S^C$
21:       $Update\ the\ LCS$
22:   end for
23: end if
    // Lines 24-30: Choose the reference solution $S^R$ from PromisingPathSet
24: $PromisingPathSet \leftarrow Local\ selection(InitialPathSet)$
25: $Select\ K-middle\ solutions\ in\ PromisingPathSet$ //Section 4.2.5.5
26: for $S^k \in K-middle\ solutions, k = \{1, \dots, N_{(KM)}\}$
     do
27:     $S^k \leftarrow Tabu\_Search(S^k)$ //Slight TS
28: end for
30: $S^R = \arg\min\{f(S^k), k = 1, \dots N_{(KM)}\}$
31: return $S^R$
```

The number of these middle solutions is defined according to the length of the *PromisingPathSet* as $N_{(KM)} = \lfloor\sqrt{l_{PPS}}\rfloor$, where l_{PPS} is the number of solutions in the *PromisingPathSet* and $N_{(KM)}$ is the smallest integer that is equal or greater than $|\sqrt{l_{PPS}}|$. Then, a slight TS is applied for optimizing *K*-middle solutions of the *PromisingPathSet* and then, the best optimized solution is selected and further optimized using a strong TS. This optimized solution is chosen as S^R. The slight TS in TS/PR determines the best neighbor solution in the CTS, without considering the OTS and ITS, while the strong TS performs the OTS and ITS for each neighbor solution generated in the CTS and OTS, respectively. The pseudo-codes for TS/PR is depicted in Table 8.

Table 8. Pseudo-code for TS/PR algorithm

```
      TS/PR Algorithm: Outline of algorithm TS/PR for HFS
1:    Input: $Parameters$ //Section 4.1.1
2:    Output: $The\ best\ objective\ function\ value$ &
                  $The\ best\ schedule\ S^{best}\ found\ so\ far$
3:    $S^{IS} \leftarrow Identify\ Initial\_Solution()$  //Section 4.2.2
4:    $S^{IMP} \leftarrow Tabu\_Search(S^{IS})$ //Section 4.2.4.6
5:    $P = \{S^1, \dots, S^{p-1}, S^{IMP}\} \leftarrow Population\_Initialization(S^{IMP})$ //Section 4.2.5.2
6:    $S^{best} = \arg\min\{f(S^l) | l = 1, \dots, p\}$
7:    $PairSet \leftarrow \{(S^i, S^j) | S^i \in P, S^j \in P\ and\ S^i \neq S^j\}$
8:    repeat
9:      $Randomly\ select\ one\ solution\ pair\ \{S^i, S^j\}$
        $from\ PairSet\ if\ \{S^i, S^j\} \notin TabuSet$
10:     $S^{p+1} \leftarrow Path\_Relinking(S^i, S^j)$,
        $S^{p+2} \leftarrow Path\_Relinking(S^j, S^i)$  //Sections 4.2.5.3
11:     $S^{p+1} \leftarrow Tabu\_Search(S^{p+1})$,
        $S^{p+2} \leftarrow Tabu\_Search(S^{p+2})$  //Strong TS
12:     if $S^{p+1}\ (or\ S^{p+2}) is\ better\ than\ S^{best}$ then
13:       $S^{best} \leftarrow S^{p+1}\ (or\ S^{p+2})$
14:     end if
15:     $Add\ S^{p+1}\ and\ S^{p+2}\ to\ population\ P: P' = P \cup \{S^{p+1}, S^{p+2}\}$
16:     $PairSet \leftarrow PairSet \cup \{(S^{p+1}, S^l) | S^l \in P\ and\ S^{p+1} \neq S^l\}$
17:     $PairSet \leftarrow PairSet \cup \{(S^{p+2}, S^l) | S^l \in P\ and\ S^{p+2} \neq S^l\}$
18:     $Identify\ the\ two\ worst\ solutions\ S^m\ and\ S^n$
        $in\ the\ temporary\ population\ P'$
19:     $Generate\ new\ population\ by\ removing\ S^n\ and\ S^m$:
        $P = \{S^1, \dots, S^p, S^{p+1}, S^{p+2}\} \setminus \{S^m, S^n\}$
20:     $Update\ PairSet$
        $PairSet \leftarrow PairSet \setminus \{(S^m, S^l) | S^l \in P\ and\ S^m \neq S^l\}$
        $PairSet \leftarrow PairSet \setminus \{(S^n, S^l) | S^l \in P\ and\ S^n \neq S^l\}$
21:     $TabuSet \leftarrow (S^i, S^j)$
22:   until $a\ stopping\ criterion\ is\ satisfied$
23:   return $S^{best}$
```

4.2.6. Particle swarm optimization

Particle Swarm Optimization (PSO) is a fast-evolutionary algorithm, which is applied on a population of candidate solutions (Eberhart and Kennedy 1995). To the best of our knowledge, this is the first time a research is being performed in the application of the PSO algorithm for a batch scheduling problem in HFS. The population of the PSO and each potential solution in the population are called *swarm* and *particle*, respectively. Each particle flies around in the multi-dimensional search space according to its current position and velocity to obtain a new position based on the two following vectors:

- The best position experienced by the particle during the previous iterations, which is called the best position (P_{best}).
- The best position experienced by all particles in the population during the previous iterations, which is called the global best position (${P_{best}}^{Global}$).

4.2.6.1. Solution representation for PSO

A solution is represented by an e-dimensional vector as the sequence of jobs within batches as well as the sequence of batches on machines, for a particular stage. Due to applying the stage-based interdependency strategy, an interaction between batch and job sequence, and unequal number of jobs assigned to batches, a solution is represented as a *pseudo matrix* in k^{th} stage. The dimensions of pseudo matrix of a particle (solution) is as follows:

$$e = \left(\sum_{i \in I^k} \sum_{s \in S_i^k} \sum_{h \in V^k} Z_{ish}^k\right) + v^k$$

$$f = \max\left(\max_{\forall\, i \in I^k \,\&\, s \in S_i^k} \sum_{j \in J_i^k} \emptyset_{isj}^k \,,\, \max_{\forall\, h \in V^k} \sum_{i \in I^k} \sum_{s \in S_i^k} Z_{ish}^k\right)$$

Therefore, a solution in k^{th} stage is presented by e vectors as a pseudo matrix, where the maximum component of vectors is f. The number of components at each row of the pseudo matrix might be unequal. The first $\left(\sum_{i \in I^k} \sum_{s \in S_i^k} \sum_{h \in V^k} Z_{ish}^k\right)$ vector of this matrix presents the sequence of jobs within batches, according to first the sequence of batches on machines and then increasing machine indices, while the last v^k vector of this matrix presents the sequence of batches on machines, according to increasing machine indices. A sequence pseudo matrix for a sequence is represented as follows:

$$S^k = \begin{bmatrix} j_{142} & j_{144} & j_{141} & \\ j_{124} & j_{123} & j_{122} & \\ j_{112} & j_{113} & j_{114} & j_{111} \\ j_{233} & j_{231} & & \\ j_{132} & j_{134} & & \\ j_{243} & j_{245} & & \\ j_{221} & & & \\ S_{14} & S_{12} & & \\ S_{11} & S_{23} & & \\ S_{13} & S_{24} & S_{22} & \end{bmatrix}$$

4.2.6.2. Algorithmic structure

In a d-dimensional search space including P'_{size} initial solutions (*swarm size*) with different e-dimensional particles, i^{th} particle in k^{th} stage is represented by pseudo matrices X_i^k and Vel_i^k as its position and velocity matrices, both with the dimension $e\times f$, i.e., $X_i^k = Vel_i^k = pseudo\ matrix_{(\mathrm{e}\times\mathrm{f})}$. d represents the maximum vectors in all particles. Each particle i flies in the search space based on its position and velocity as follows:

$$X_i^k = \begin{bmatrix} X_{i11}^k & \dots\dots\dots & X_{i1f}^k \\ \vdots & \ddots & \vdots \\ X_{i(e-v^k)1}^k & \cdots & X_{i(e-v^k)f}^k \\ X_{i(e-v^k+1)1}^k & \cdots & X_{i(e-v^k+1)f}^k \\ \vdots & \ddots & \vdots \\ X_{ie1}^k & \cdots & X_{ief}^k \end{bmatrix} \begin{array}{l} \rightarrow \text{the job sequence within the first batch assigned to the first machine in } k^{th} \text{ stage} \\ \vdots \\ \rightarrow \text{the job sequence within the last batch assigned to the last machine in } k^{th} \text{ stage} \\ \rightarrow \text{the batch sequence assigned to the first machine in } k^{th} \text{ stage} \\ \vdots \\ \rightarrow \text{the batch sequence assigned to the last machine in } k^{th} \text{ stage} \end{array}$$

$$Vel_i^k = \begin{bmatrix} Vel_{i11}^k & \dots\dots\dots & Vel_{i1f}^k \\ \vdots & \ddots & \vdots \\ Vel_{i(e-v^k)1}^k & \cdots & Vel_{i(e-v^k)f}^k \\ Vel_{i(e-v^k+1)1}^k & \cdots & Vel_{i(e-v^k+1)f}^k \\ \vdots & \ddots & \vdots \\ Vel_{ie1}^k & \cdots & Vel_{ief}^k \end{bmatrix}$$

The velocity and position update equations associated with each component of each vector in each particle are calculated as follows:

$$Vel_{ijl}^{(t+1)^k} = \omega Vel_{ijl}^{(t)^k} + c_1 r_1 \left(Y_{ijl}^k - X_{ijl}^{(t)^k}\right) + c_2 r_2 \left(\hat{Y}_{jl}^k - X_{ijl}^{(t)^k}\right) \tag{4.69}$$

$$\chi \times Vel_{ijl}^{(t+1)^k} \rightarrow Vel_{ijl}^{(t+1)^k} \tag{4.70}$$

$$X_{ijl}^{(t+1)^k} = Vel_{ijl}^{(t+1)^k} + X_{ijl}^{(t)^k} \tag{4.71}$$

where $i = 1,2,\dots,P'_{size}; j = 1,2,\dots,e; l = 1,2,\dots,f; t = 1,2,\dots,Itr_{Max}; k \in K$. $X_{ijl}^{(t)^k}$ and $Vel_{ijl}^{(t)^k}$ are the position and velocity, respectively, of the component corresponding to the j^{th} row and l^{th} column of i^{th} particle at iteration t of PSO in k^{th} stage. Itr_{Max} stands for maximum iteration number of PSO. Y_{ijl}^k and $\hat{Y}_{jl}^k$ are P_{best} and ${P_{best}}^{Global}$, respectively. ω is the inertia weight to determine the impact of the previous velocity of the particle in the next iteration. The constant values of c_1 and c_2, known as acceleration coefficients, determine the impact of P_{best} and ${P_{best}}^{Global}$ to define velocity, respectively, while r_1 and r_2 are incorporated in velocity to consider uncertainty in the meta-heuristic algorithm. In order to control the extreme roaming of particles outside of the search space, the new velocity and position value are restricted to the interval $[vel^{min}, vel^{max}]$ and $[x^{min}, x^{max}]$, respectively. Generally, the new velocities of particles are determined by Eq. (4.69) according to their previous velocities as well as the distance of their current position from both P_{best} and ${P_{best}}^{Global}$. In order to improve the performance of the PSO algorithms, a multiplier χ in Eq. (4.70) is implemented to accelerate the process of the convergence (Poli et al. 2007). Then, the new position of each particle is determined by Eq. (4.71) according to its new velocity and previous position. In ${Itr_{Max}}^{th}$ iteration of PSO, ${P_{best}}^{Global}$ is reported.

4.2.6.3. Encoding and decoding of particles

A conversion on the components of the sequence pseudo matrix to the continuous position values, i.e., ${S_i^{(t+1)}}^k \rightarrow {X_i^{(t+1)}}^k$, must be performed to apply the PSO algorithm. ${S_i^{(t)}}^k$ represents the sequence pseudo matrix of i^{th} particle at iteration t of PSO in k^{th} stage. The following encoding scheme converts job and batch sequences to the continuous position values.

$$j_{sij} \longrightarrow \left(\left(\sum_{x=0}^{i-1} \sum\nolimits_{y \in S_i^k} \sum\nolimits_{j \in J_i^k} \emptyset_{xyj}^k \right) + \left(\sum_{y=0}^{s-1} \sum\nolimits_{j \in J_i^k} \emptyset_{iyj}^k \right) + \left(\sum_{z=1}^{j-1} \emptyset_{isz}^k \right) \right); \; i \in I^k; \; s \in S_i^k; j \in J_i^k$$

$$S_{si} \longrightarrow \left(\left(\sum_{x=0}^{i-1} \sum\nolimits_{y \in S_i^k} \sum\nolimits_{h \in V^k} Z_{xyh}^k \right) + s \right); \; i \in I^k; \; s \in S_i^k$$

where $\emptyset_{xyj}^k$ is equal to zero for each $x = 0$ and/or $y = 0$. Also, Z_{xyh}^k is equal to zero when $x = 0$. These transformations convert the initial sequence pseudo matrix determined in section 4.2.6.1 to the following pseudo matrix:

$$X^k = \begin{bmatrix} 14 & 15 & 13 & \\ 7 & 6 & 5 & \\ 2 & 3 & 4 & 1 \\ 12 & 11 & & \\ 9 & 10 & & \\ 16 & 17 & & \\ 8 & & & \\ 6 & 2 & & \\ 1 & 5 & & \\ 4 & 7 & 3 & \end{bmatrix}$$

A decoding scheme based on Ranked Order Value (ROV) shown in Table 9 is developed to convert the continuous position value of a particle to job and batch sequences, i.e., ${X_i^{(t+1)}}^k \rightarrow {S_i^{(t+1)}}^k$, for each iteration of the PSO algorithm. In the ROV rule, the Smallest Position Value (SPV) technique along with one-to-one correspondence between ${X_i^{(t+1)}}^k$ and ${S_i^{(t)}}^k$(meaning ${X_{i11}^{(t+1)}}^k$ with ${S_{i11}^{(t)}}^k$, and so on to finally consider ${X_{ief}^{(t+1)}}^k$ with ${S_{ief}^{(t)}}^k$) are used for this transformation, i.e, ${X_i^{(t+1)}}^k \rightarrow {S_i^{(t+1)}}^k$.

Table 9. Pseudo-code for decoding a position pseudo matrix to a sequence pseudo matrix

```
**for each** $i \in \{1,2,\dots,P'_{size}\}$ **do**
    **for each** $j \in \{1,2,\dots,e\}$ **do**
        ${S_{ij}^{(t+1)}}^k \leftarrow {X_{ij}^{(t+1)}}^k$ //According to the SPV rule in ${X_i^{(t+1)}}^k$ as well as ${S_i^{(t)}}^k$
    **end for**
**end for**
```

For example, ${S_{i1}^{(t+1)}}^k = (j_{213}, j_{211}, j_{215}, j_{216})$, when ${S_{i1}^{(t)}}^k = (j_{211}, j_{215}, j_{213}, j_{216})$ and ${X_{i1}^{(t+1)}}^k = (2.98, 3.04, 1.28, 3.92)$. The smallest value in ${X_{i1}^{(t+1)}}^k$ is 1.28, which is located at the third position in vector ${X_{i1}^{(t+1)}}^k$. Therefore, the job in the third position of ${S_{i1}^{(t)}}^k$ should be the first job in ${S_{i1}^{(t+1)}}^k$ (i.e., ${S_{i11}^{(t+1)}}^k = j_{213}$). Other components in ${S_{i1}^{(t+1)}}^k$ are determined using a similar procedure.

4.2.7. Particle swarm optimization/local search

The lack of mechanism for batching becomes more pronounced when a basic PSO algorithm is applied for batch scheduling. The neighborhood searches for the best batch sequence on machines as well as the best job sequence within batches are *naturally* performed by updating particles, in each iteration of PSO, i.e., the scheduling phase. Therefore, a PSO algorithm is accompanied by a local search algorithm (LSA) between each of two sequential iterations of PSO (PSO/LSA) to take the benefits of batching and, consequently, enhance the quality of solutions, i.e., the batching phase. The LSA is performed at two search levels as follows:

- After converting S_i^k to its own CSD^k and OSD^k (Section 4.2.4.3), a feasible neighbor solution in the CSD^k is generated by only insertion-related operator in the first search level.
- Then in the second search level, for each generated neighbor solution better than the current seed, the best feasible neighbor solution(s) of the OSD^k are determined by both insertion- and swap-related operator.
- The best neighbor solution obtained by both search levels of the LSA is considered as the next seed and the process repeats until a stopping criterion is reached.

The pseudo-code for developed PSO/LSA is presented in Table 10. It is worth noting that the difference between the local search structure in PSO/LSA and TS/PR is that the LSA is applied to enhance the performance of PSO at two levels, while TS is naturally a local search algorithm, which is performed at three levels and accompanied by a PR procedure to integrate diversification and intensification strategies of TS. If there is not any feasible neighbor solution in a level of the LSA, the best infeasible neighbor solution should be converted to a feasible one with the help of the adjustment step. Since there is a need to keep track of the best position of batches during all previous iterations of PSO/LSA, the refinement step should not be applied for selected neighbor solutions during the LSA.

Table 10. Pseudo-code for PSO/LSA algorithm

```
    PSO/LSA Algorithm: Outline of algorithm PSO/LSA for HFS
1:  Input: $Parameters$ //Section 4.1.1
2:  Output: $The\ best\ objective\ function\ value$ & $The\ best\ schedule\ S^{best}\ found\ so\ far$
3:  $IP \leftarrow Population.Initialization()$ //Pseudo-code in section 4.2.7
4:  $t = 0$
5:  while $t \leq Itr_{max}$ do
        // Lines 6-10: Apply PSO
6:      for each $i \in \{1,2,\dots,P'_{size}\}$ do
7:          $Vel_{ijl}^{(t+1)^k} \leftarrow \omega Vel_{ijl}^{(t)^k} + c_1 r_1 \left(Y_{ijl}^k - X_{ijl}^{(t)^k}\right) + c_2 r_2 \left(\hat{Y}_{jl}^k - X_{ijl}^{(t)^k}\right)$
8:          $Vel_{ijl}^{(t+1)^k} \leftarrow \chi \times Vel_{ijl}^{(t+1)^k}$
9:          $X_{ijl}^{(t+1)^k} \leftarrow Vel_{ijl}^{(t+1)^k} + X_{ijl}^{(t)^k}$
10:     end for
        // Lines 11-23: Enhance solutions with a local search algorithm (LSA)
11:     for each $i \in \{1,2,\dots,P'_{size}\}$ do
12:         $S_i^{(t+1)^k} \leftarrow X_i^{(t+1)^k}$ //According to the SPV rule in $X_i^{(t+1)^k}$ as well as $S_i^{(t)^k}$
13:         $CSD^{S_i^{(t+1)^k}} \leftarrow S_i^{(t+1)^k}$ & $OSD^{S_i^{(t+1)^k}} \leftarrow S_i^{(t+1)^k}$
14:         $t' = 0$
15;         while $t' \leq Itr_{LSA}$ do
16:             $Perform\ First.Search.Level.LSA(CSD^{S_i^k})$
17:             $Perform\ Second.Search.Level.LSA(OSD^{S_i^k})$
18:         end while
19:         $S_i^{(t+1)^k} \leftarrow CSD^{S_i^{(t+1)^k}}$
20:         $S_i^{(t+1)^k} \leftarrow Update.S_i^{(t+1)^k}()$ //Section 4.2.7.1
21:         $X_i^{(t+1)^k} \leftarrow Update.X_i^{(t+1)^k}()$ //Section 4.2.7.1
22:         $Y_i^k \leftarrow Update.Y_i^k()$ //$P_{best}$ of $i^{th}$ particle
23:     end for
24:     $\hat{Y}^k \leftarrow Update.\hat{Y}^k()$ //${P_{best}}^{Global}$ related to all particles
25: end while
26: return $S^{best} \leftarrow {P_{best}}^{Global}$
```

A graphical example of one iteration of the two-level LSA is illustrated in Figure 17. In this example, for four neighbor solutions chosen in the first search level, the second search level is applied and then the solution with the objective function value 3431.03 is replaced by the current seed (3982.80).

In order to maintain consistency between TS/PR and PSO/LSA, both algorithms select the required IP based on similar procedure. The pseudo-code shown in Table 11 determines the IP with P'_{size} members.

Table 11. Pseudo-code for initializing PSO population

```
$S^{IS} \leftarrow Identify\ Initial.Solution()$ //Section 4.2.2
$S^{IMP} \leftarrow LSA(S^{IS})$ //Section 4.2.7
$IP \leftarrow Encoding(S^{IMP})$
for each $i \in \{1,2,\dots,P'_{size}-1\}$ do
    $Select\ a\ NonRepeated\ random\ solution\ S_i^k\ from\ the\ best\ solutions\ of\ LSA_2$
    $IP \leftarrow Encoding(S_i^k)$
    if $i = LSA_{2_{SIZE}}$ then
        exit for
end for
if $P'_{size} > LSA_{2_{SIZE}} + 1$ then
    for each $i \in \{1,2,\dots,P'_{size} - LSA_{2_{SIZE}} - 1\}$ do
        $Select\ a\ NonRepeated\ random\ solution\ S_i^k\ from\ the\ best\ solutions\ of\ LSA_1$
        $IP \leftarrow Encoding(S_i^k)$
        if $i = LSA_{1_{SIZE}}$ then
            exit for
    end for
    if $P'_{size} > LSA_{1_{SIZE}} + LSA_{2_{SIZE}} + 1$ then
        for each $i \in \{1,2,\dots,P'_{size} - LSA_{1_{SIZE}} - LSA_{2_{SIZE}} - 1\}$ do
            $vel_{ij}^{(0)} = vel^{min} + R(vel^{max} - vel^{min})$
            $x_{ij}^{(0)} = x^{min} + R(x^{max} - x^{min})$ // $R \in unif[0,1]$
            if $x_{ij}^{(0)}$ violates the machine eligibilities and/or $LB_{ih}^k$ then
                $i = i - 1$
                $continue\ for$
            else
                $IP \leftarrow x_{ij}^{(0)}$
            end if
        end for
    end if
end if
// $LSA_1$ & $LSA_2$: the first and second search level of the LSA
```

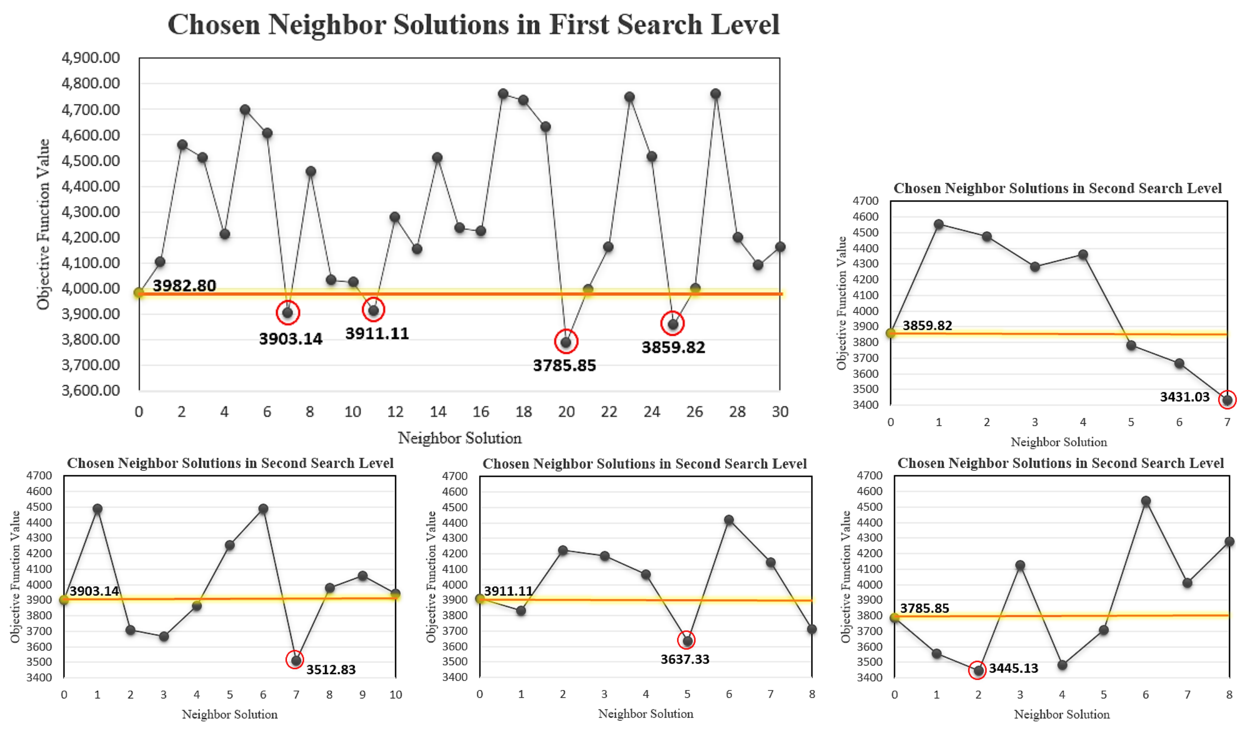


Figure 17. Illustration for two-level local search algorithm (LSA)

4.2.7.1. Updating process

The following three-phase updating process is implemented for modifying batch compositions of $OSD^{S_i^k}$ after completing the LSA, updating S_i^k and, subsequently, X_i^k before passing them to the next iteration of PSO/LSA, and updating the values of Y_{ijl}^k and $\hat{Y}_{jl}^k$ for each iteration of PSO/LSA.

***Updating S_i^k*:** after completing the LSA, two sequential batches of i^{th} group ($\forall i \in I^k$) in the $OSD^{S_i^k}$ ($\forall i \in \{1,2,\dots,P'_{size}\}$) should be merged as a single batch with the batch number equal to the batch number of S_{si} including more jobs between two sequential batches. Ties are broken in favor of the smaller batch number. A batch number is referred to sub-index s of S_{si}. Apart from this, after splitting a batch of i^{th} group into two batches in the $OSD^{S_i^k}$, the batch including more jobs will have the same batch number of its parent, while the batch number of another batch is determined as the available smallest batch number in the series of batch numbers, i.e., from 1 up to n_i^k, so that this batch number is not equal to the existing batch numbers of i^{th} group. A result of this step is updated S_i^k in terms of new batch compositions and batch assignment by updating $CSD^{S_i^k}$ with regard to $OSD^{S_i^k}$ and then transforming $CSD^{S_i^k}$ to S_i^k.

***Updating X_i^k*:** after completing the LSA and, consequently, updating S_i^k, the continuous position X_i^k related to updated S_i^k should be updated (i.e., updated X_i^k) to guarantee that the sequence resulting from the ROV rule for the new continuous position is the same as the sequence resulting from the two-level LSA. This being the case, the position value of each member of updated S_i^k, i.e., updated X_{ijl}^k, is obtained by one-to-one correspondence between X_i^k and S_i^k (i.e., S_i^k before updating process) in terms of updated S_{ijl}^k. As a result, the position values of jobs and batches related to updated S_i^k, i.e., updated X_{ijl}^k, and subsequently, updated X_i^k are obtained. The position value of a batch with the new batch number is determined by previous encoding rule developed in section 4.2.6.3.

***Updating P_{best} & $P_{best}{}^{Global}$*:** due to creating different batch compositions during each iteration of PSO/LSA, P_{best} and $P_{best}{}^{Global}$ of a job cannot depend on its position in a particular batch. Therefore, the assignment and position of a job on a machine are considered as P_{best} and $P_{best}{}^{Global}$ for the job. In other words, the values of Y_{ijl}^k and $\hat{Y}_{jl}^k$ are updated based on the position of jobs on machines, instead of batches. Likewise, the assignment and position of a batch on a machine are considered as P_{best} and $P_{best}{}^{Global}$ for the batch.

4.2.8. Ineffective neighbor moves

In addition to developing specific neighborhood structures, we present a *method* to reduce the computational burden of a local search algorithm by identifying neighborhoods, which have either no effect or an inferior effect on the objective function value. These ineffective neighborhoods determined by the lemmas, presented below, are usually not a candidate for a move. Applying these lemmas guarantee obtaining the same optimal/near optimal solution by the search algorithm in less computational time. Either all or part of these lemmas are applicable for any local search structure of developed algorithms. Prior to identifying ineffective neighbor moves based upon specific lemmas in terms of both insertion- and swap-related moves, we first introduce the notations relating to a forbidden candidate move.

The following notations are applicable only for lemmas. Consider a particular stage of HFS. Suppose j_k^m and $j_{k'}^{m'}$ represent job j processed on the k^{th} position of m^{th} machine in the initial schedule and the k'^{th} position of m'^{th} machine in the new schedule, respectively, irrespective of which batch the job belongs to. $m,\ m' \in [1,\dots,m_a,m_b,\dots,M]$, where M is the number of unrelated-parallel machines for the selected stage, i.e., $M = v^k, \forall\ k \in m$. The " ′ " sign applies to the new schedule. There are n and n' jobs on m^{th} and m'^{th} machines, respectively, meaning $k \in [1,2,\dots,n]$ and $k' \in [1,2,\dots,n']$. As a result, for two different machines m_a and m_b, the number of jobs assigned to these machines are different in the new schedule compared to the initial schedule by inserting one job from m_a to m_b (i.e., $n'_a = n_a - 1$ and $n'_b = n_b + 1$) or vice versa. The k^{th} and k'^{th} positions of any job j are not necessarily equal on the same machine ($m = m'$), meaning $k' = k$ for job j when its position is not changed, $k' > k$ ($k' \in [k+1,n]$) for job j

inserted/swapped forwardly, $k' = k + 1$ for job j pushed forwardly, $k' < k$ ($k' \in [1, k-1]$) for job j inserted/swapped backwardly, and $k' = k - 1$ for job j pushed backwardly. Likewise, the k^{th} and k'^{th} positions of any job j inserted/swapped are not necessarily equal on different machines ($m \neq m'$), meaning $k' \in [1, \ldots, k, \ldots, n']$.

- **Definition 1 (*Forward Move*):** By applying a forward move (insert or swap), job j_k^m can be inserted into the k'^{th} position between jobs across batches on the same machine m, so that $k' > k$ ($k' \in [k+1, n]$).
- **Definition 2 (*Backward Move*):** By applying a backward move (insert or swap), job j_k^m can be inserted into the k'^{th} position between jobs across batches on the same machine m, so that $k' < k$ ($k' \in [1, k-1]$).

In the initial schedule, the completion time of job j_k^m ($CT_{j_k^m}$) is equal to the start time of this job ($ST_{j_k^m}$) plus its run time ($RT_{j_k^m}$). The start time of job j_k^m is the maximum time between its release time ($r_{j_k^m}$), and the completion time of job j_{k-1}^m ($CT_{j_{k-1}^m}$) plus the required setup time ($S_{j_{k-1}^m j_k^m}$) between jobs assigned to k^{th} and $(k-1)^{th}$ positions on the same machine m (i.e., $ST_{j_k^m} = \max\{r_{j_k^m}, CT_{j_{k-1}^m} + S_{j_{k-1}^m j_k^m}\}$), regardless of which machine j_k^m will be assigned/inserted to. Likewise, in the new schedule, the start time of $j_{k'}^{m'}$ is $ST_{j_{k'}^{m'}} = \max\{r_{j_k^m}, CT_{j_{k'-1}^{m'}} + S_{j_{k'-1}^{m'} j_{k'}^{m'}}\}$. Note that the release time of a job, j_k^m, does not relate to both the machine which it is assigned/inserted to and its position on the machine in both new and initial schedules, but for clarity, the release time of a job is represented as $r_{j_k^m}$.

$CT_{j_{k'}^{m'}}$ is not changed, regardless of which machine $j_{k'}^{m'}$ will be inserted/swapped (i.e., $m = m'$ or $m \neq m'$), when $ST_{j_{k'}^{m'}} = ST_{j_k^m}$, meaning either $r_{j_k^m} = ST_{j_k^m}$ or $CT_{j_{k-1}^m} + S_{j_{k-1}^m j_k^m} = CT_{j_{k'-1}^{m'}} + S_{j_{k'-1}^{m'} j_{k'}^{m'}}$. Apart from this, $CT_{j_{k'}^{m'}}$ is increased, regardless of which machine $j_{k'}^{m'}$ will be inserted/swapped, when $ST_{j_{k'}^{m'}} > ST_{j_k^m}$, meaning $CT_{j_{k'-1}^{m'}} + S_{j_{k'-1}^{m'} j_{k'}^{m'}} > CT_{j_{k-1}^m} + S_{j_{k-1}^m j_k^m}$. The above inequality may be satisfied, except when $CT_{j_{k'-1}^{m'}} < CT_{j_{k-1}^m}$ and $S_{j_{k'-1}^{m'} j_{k'}^{m'}} < S_{j_{k-1}^m j_k^m}$. As a result, the $CT_{j_{k'}^{m'}}$ can be equal to or greater than the $CT_{j_k^m}$, when $CT_{j_{k'-1}^{m'}} < CT_{j_{k-1}^m}$, $S_{j_{k'-1}^{m'} j_{k'}^{m'}} > S_{j_{k-1}^m j_k^m}$, and the increase in $S_{j_{k'-1}^{m'} j_{k'}^{m'}}$ is big enough to compensate for the reduction in $CT_{j_{k'-1}^{m'}}$, so that $CT_{j_{k'-1}^{m'}} + S_{j_{k'-1}^{m'} j_{k'}^{m'}}$ is equal to or greater than $CT_{j_{k-1}^m} + S_{j_{k-1}^m j_k^m}$. Likewise, $CT_{j_{k'}^{m'}} \geq CT_{j_k^m}$, when $CT_{j_{k'-1}^{m'}} > CT_{j_{k-1}^m}$, $S_{j_{k'-1}^{m'} j_{k'}^{m'}} < S_{j_{k-1}^m j_k^m}$, and the increase in $CT_{j_{k'-1}^{m'}}$ is big enough to compensate for the reduction in $S_{j_{k'-1}^{m'} j_{k'}^{m'}}$, so that $CT_{j_{k'-1}^{m'}} + S_{j_{k'-1}^{m'} j_{k'}^{m'}}$ is equal to or greater than $CT_{j_{k-1}^m} + S_{j_{k-1}^m j_k^m}$.

In conclusion, the completion time of a job is either increased or not changed in the new schedule, *if and only if* the start time of this job in the new schedule is either increased or not changed compared to the one in the initial schedule ($ST_{j_{k'}^{m'}} \geq ST_{j_k^m}$). A move is ineffective when the completion time of jobs, both inserted/swapped and assigned to the machine(s), are either increased or not changed by applying this move. Sometimes, the completion time of a job is not changed in spite of shifting forwardly the job(s) processed before this job on a machine due to the existence of idle times on the same machine. The idle time can be created on a machine when the machine availability time is less than the release time of remaining jobs, which have not been processed so far by this machine.

Only the jobs assigned to a machine/machines that are considered for performing a predetermined insert or swap move may have an effect on changing the objective function value (ΔZ). Among these jobs, the job $j_{k'}^{m'}$ always has no contribution to ΔZ, when $CT_{j_{k'-1}^{m'}} = CT_{j_{k-1}^m}$ and $S_{j_{k'-1}^{m'} j_{k'}^{m'}} = S_{j_{k-1}^m j_k^m}$. In other words, only the jobs that are assigned to this machine(s) and have a contribution in the objective function value should be considered in evaluating ΔZ. The variation in the completion time of a job has a forward relationship with its tardiness. In other words, by increasing the completion time of a job, the related tardiness is either increased or not changed. Likewise, by decreasing the completion time of a job, the related tardiness is either decreased or not changed. It is clear that the tardiness of a job does not change while its completion time is not changed.

Note that the reference group should be considered as the initial batch when there is no job/batch before the new position of the job/batch moved backwardly in the new schedule, or the job/batch processed immediately before the job/batch moved forwardly in the initial schedule. In all of the following lemmas, the completion time of jobs is either increased or not changed by applying predetermined insert and swap moves.

Lemma 1-1: *Regardless of which batch of the same group or a different group job j_{k-1}^m will be inserted to, due date and release time of other jobs, applying a forward inserting move for job j_{k-1}^m on the same machine provides no improvement in the objective function value, if*

I. $CT_{j_{k'}^{m'}} \geq CT_{j_k^m}$ *, where $k' = k - 1$ and $m = m'$; and*

II. $CT_{j_{(k'+1)'}^{m'}} \geq CT_{j_{(k')}^m}$ *, where $k' \in [k, n]$ and $m = m'$.*

Proof: Note that by inserting forwardly job j_{k-1}^m at any available position on the same machine, the k'^{th} position of job j in the new schedule is greater than its $(k-1)^{th}$ position in the initial schedule ($k' > k -$

1 $(k' \in [k, n])$). By doing this, the completion time of all jobs assigned to the same machine in the new schedule are either increased or not changed as follows:

- The $CT_{j_{l'}^m}$ does not change, where $l' \in [1, k-2]$
- The $CT_{j_{l'}^m}$ is either increased or not changed, where $l' \in [k-1, k'-1]$
- The $CT_{j_{k'}^m}$ increases, and
- The $CT_{j_{l'}^m}$ is either increased or not changed, where $l' \in [k'+1, n]$.

Therefore, the minimum increase in ΔZ due to the change in the completion time of job j inserted forwardly and its tardiness can be evaluated as $\Delta Z \geq \ w_j \left(\alpha\left(CT_{j_{k'}^m} - CT_{j_{k-1}^m}\right) + \ \beta\left(T_{j_{k'}^m} - T_{j_{k-1}^m}\right)\right)$, where w_j is the weight of job j, α and β are the weights attributed to the producer and customers, respectively, and $T_{j_{k'}^m}$ and $T_{j_{k-1}^m}$ are the tardiness of job j assigned to the k'^{th} and $(k-1)^{th}$ positions in the new and initial schedules, respectively. The tardiness of j_k^m is equal to $\max\{0, CT_{j_k^m} - d_j\}$. If the due date of job j (d_j) is equal to or greater than its completion time in the new schedule $(d_j \geq CT_{j_k^m})$, the difference in the objective function value can be written as $\Delta Z \geq \ w_j \left(\alpha\left(CT_{j_{k'}^m} - CT_{j_{k-1}^m}\right)\right)$.

Lemma 1-2: *Regardless of which batch of the same group or a different group job j_k^m will be inserted to, due date and release time of other jobs, applying a backward inserting move for job j_k^m on the same machine provides no improvement in the objective function value, if*

I. $CT_{j_{k'}^{m'}} \geq \ CT_{j_k^m}$ *, where $k' \in [1, k-1]$ and $m = m'$; and*

II. $CT_{j_{k'}^{m'}} \geq \ CT_{j_{k+1}^m}$ *, where $k' = k+1$ and $m = m'$.*

Proof: Note that by inserting backwardly job j_k^m at any available position on the same machine, the k'^{th} position of the job j in the new schedule is less than its k^{th} position in the initial schedule ($k' < k$ ($k' \in [1, k-1]$)). By doing this, the completion time of all jobs assigned to the same machine in the new schedule are either increased or not changed as follows:

- The $CT_{j_{l'}^m}$ does not change, where $l' \in [1, k'-1]$
- The $CT_{j_{k'}^m}$ is either increased or not changed

- The $CT_{j_{l'}^m}$ is increased, where $l' \in [k'+1, k]$ (referred to as common job(s)), and
- The $CT_{j_{l'}^m}$ is either increased or not changed, where $l' \in [k+1, n]$.

Therefore, the minimum increase in ΔZ due to the change in the completion time of common job(s) and its tardiness can be evaluated as $\Delta Z \geq \sum_{l\in[k',k-1],l'\in[k'+1,k]} w_j \left(\alpha\left(CT_{j_{l'}^m} - CT_{j_l^m}\right) + \beta\left(T_{j_{l'}^m} - T_{j_l^m}\right)\right)$ by considering one-to-one correspondence between l and l' (meaning k' with $k'+1$, and so on to finally consider $k-1$ with k), which includes the common job(s), which are processed after $j_{k'}^m$ in the new schedule and before j_{k+1}^m in the initial schedule. If the due date of the common job(s) assigned to $[j_{k'+1}^m{}'\, j_k^m]$ is equal to or greater than its completion time on the new schedule, the difference in the objective function value can be written as $\Delta Z \geq \sum_{l\in[k',k-1],l'\in[k'+1,k]} w_j \left(\alpha\left(CT_{j_{l'}^m} - CT_{j_l^m}\right)\right)$.

Lemma 1.1 and 1.2 are illustrated in Figure 18. The first forward insertion-related-move illustrates inserting job i after job j on the same machine. The completion times of jobs processed after job j in developed neighborhood are either increased or not changed (for the rest of schedule), because the release time of job j in the developed neighborhood freezes the schedule for backward movement. By assuming job i and j belong to the s^{th} batch of group g, move $v_I(S_{sg}|1,2)$ is stored in the ITL.

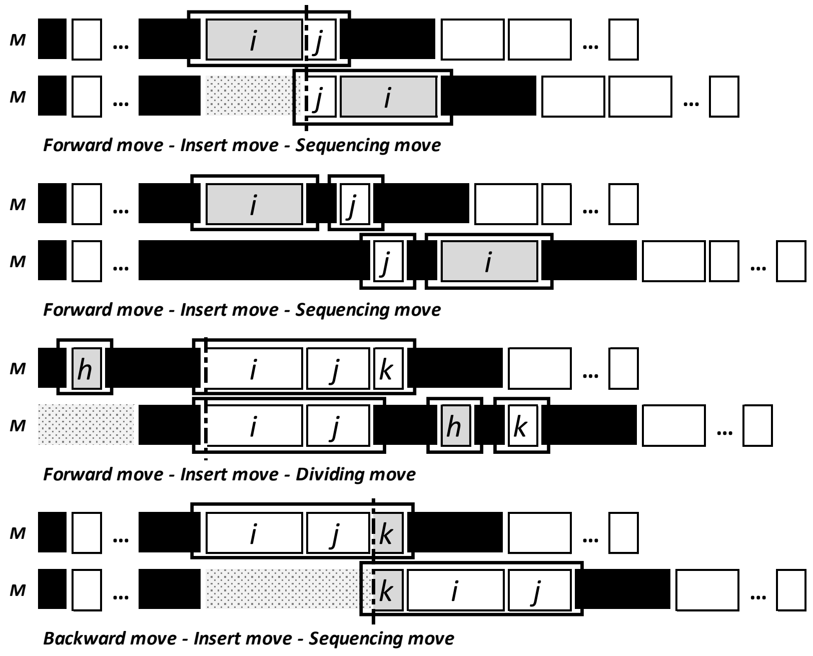

M
i
j
Forward move - Insert move - Sequencing move
Forward move - Insert move - Sequencing move
h
k
Forward move - Insert move - Dividing move
Backward move - Insert move - Sequencing move

Figure 18. Illustration for lemma 1-1 & 1-2

The second forward insertion-related-move is the same as the first forward move, while instead of release time, the new setup time related to job j in the developed neighborhood freezes job j for backward movement. If job i is assigned to position p on machine M, move $v_I(M_p, M_{p+1})$ is stored in the OTL. The third forward insertion-related-move is again the same as the first forward move and the release time of job i freezes the schedule. In addition, this move changes the batch composition and represents a dividing move. In this case, move $v_I(M|g_g, g_{g'}, g_g)$ is stored in the CTL, in which job h belongs to group g' and job j and k belong to group g. Finally, the forth backward insertion-related-move illustrates inserting job k before job i on the same machine. Likewise, job k in the developed neighborhood freezes the schedule due to its release time. The first, second, and forth moves are considered as sequencing move because they only change the sequence of jobs on batches/machines.

Lemma 2: *Regardless of which batch of the same group or a different group job $j_{k-1}^{m_a}$ will be inserted to, due date and release time of other jobs, inserting job $j_{k-1}^{m_a}$ in any available position between jobs across batches on a different machine (m_b) provides no improvement in the objective function value, if*

I. $CT_{j_{k'}^{m'_a}} \geq CT_{j_k^{m_a}}$ *, where* $k' = k - 1$ *and* $m'_a = m_a$ *; and*

II. $CT_{j_{(k'+1)'}^{m'_a}} \geq CT_{j_{(k')}^{m_a}}$ *, where* $k' \in [1, n_b]$ *and* $m'_a = m_a$ *; and*

III. $CT_{j_{k'}^{m'_a}} \geq CT_{j_{k-1}^{m_a}}$ *, where* $k' \in [1, n_b]$ *and* $m'_a = m_b$ *.*

Proof: Note that by inserting job $j_{k-1}^{m_a}$ on a different machine m_b, the k'^{th} position of the job j in the new schedule can be any available position as $k' \in [1, n_b]$. By doing this, the completion time of all jobs which are assigned to machines m_a and m_b in the new schedule are either increased or not changed as follows:

- The $CT_{j_{l'}^{m'_a}}$ does not change, where $l' \in [1, k-2]$ and $m'_a = m_a$
- The $CT_{j_{l'}^{m'_a}}$ is either increased or not changed, where $l' \in [k-1, n_a - 1]$ and $m'_a = m_a$
- The $CT_{j_{l'}^{m'_b}}$ does not change, where $l' \in [1, k'-1]$ and $m'_b = m_b$
- The $CT_{j_{k'}^{m'_a}}$ increases or does not change, where $m'_a = m_b$ and $k' \in [1, n_b]$, and
- The $CT_{j_{l'}^{m'_b}}$ is either increased or not changed, where $l' \in [k'+1, n_b + 1]$ and $m'_b = m_b$.

The minimum increase in ΔZ is positive when at least the completion time of one job belonging to either $[k-1, n_a-1]$ of machine m_a or $[k', n_b+1]$ of machine m_b in the new schedule is increased; otherwise, the minimum increase in ΔZ is equal to zero.

Lemma 2 is illustrated in Figure 19. The insertion-related-move illustrates inserting job i, which is assigned to machine M_1 before job k, which is assigned to machine M_2. The completion times of jobs processed after job i and j in developed neighborhood are either increased or not changed (for the rest of schedule), because the release time of job i and j in the developed neighborhood freezes the schedule for backward movement on machines M_1 and M_2, respectively. In this case, move $v_I(M_2|g_{g'}, g_g, g_g)$ is stored in the CTL, in which job i, j, k and h belong to group g. Group g' belongs to the job, which is processed immediately before job i on machine M_2.

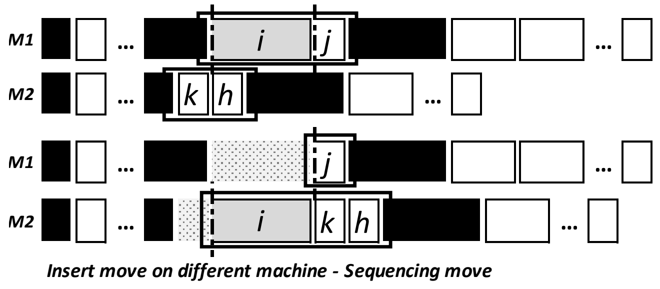


Figure 19. Illustration for lemma 2

Lemma 3.1: *Regardless of which batches of the same group or different groups jobs $j^m_{k_a}$ and $j^m_{k_b}$ will be swapped, due date and release time of other jobs, swapping two jobs $j^m_{k_a}$ and $j^m_{k_b}$ on the same machine, where $k_b > k_a$ provides no improvement in the objective function value, if*

I. $CT_{j^{m'}_{k'_b}} \geq CT_{j^m_{k_b}}$ *, where $k'_b = k_a$ and $m' = m$; and*

II. $CT_{j^{m'}_{k'_b}} \geq CT_{j^m_{k_b+1}}$ *, where $k'_b = k_b + 1$ and $m' = m$.*

Proof: Note that by swapping job $j^m_{k_b}$ with job $j^m_{k_a}$ processed before $j^m_{k_b}$ on the same machine, the k'^{th} positions of jobs $j^{m'}_{k'_b}$ and $j^{m'}_{k'_a}$ in the new schedule is k_a and k_b, respectively. By doing this, the completion time of all jobs assigned to machine m in the new schedule, are either increased or not changed as follows:

- The $CT_{j^m_{l'}}$ does not change, where $l' \in [1, k_a - 1]$
- The $CT_{j^m_{k'_b}}$ increases or does not change

- The $CT_{j_{l'}^{m}}$ is increased, where $l' \in [k_a + 1, k_b - 1]$
- The $CT_{j_{k'_a}^{m}}$ increases, and
- The $CT_{j_{l'}^{m}}$ is either increased or not changed, where $l' \in [k_b, n]$.

The minimum increase in ΔZ due to the change in the completion time of job(s) assigned to $[k_a + 1, k_b]$ in the new schedule and their tardiness can be evaluated as $\Delta Z \geq \sum_{l \in [k_a, k_b - 1], l' \in [k_a + 1, k_b]} w_j \left(\alpha \left(CT_{j_{l'}^{m}} - CT_{j_{l}^{m}}\right) + \beta \left(T_{j_{l'}^{m}} - T_{j_{l}^{m}}\right)\right)$ by considering one-to-one correspondence between l and l' (meaning k_a with $k_a + 1$, and so on to finally consider $k_b - 1$ with k_b). If the due date of the jobs assigned to $[j_{(k'_b+1)}^{m} \, ' \, j_{k'_a}^{m}]$ is equal to or greater than their completion times in the new schedule, then $\Delta Z \geq \sum_{l \in [k_a, k_b - 1], l' \in [k_a + 1, k_b]} w_j \left(\alpha \left(CT_{j_{l'}^{m}} - CT_{j_{l}^{m}}\right)\right)$.

Lemma 3-2: *Regardless of which batches of the same group or different group jobs $j_{k_a}^{m_a}$ and $j_{k_b}^{m_b}$ will be swapped, due date and release time of other jobs, swapping two jobs $j_{k_a}^{m_a}$ and $j_{k_b}^{m_b}$ on different machines ($m_a \neq m_b$) provides no improvement in the objective function value, if*

I. $CT_{j_{k'_a}^{m'_a}} \geq CT_{j_{k_a}^{m_a}}$ *, where $m'_a = m_b$ and $k'_a = k_b$; and*

II. $CT_{j_{k'_b}^{m'_b}} \geq CT_{j_{k_b}^{m_b}}$ *, where $m'_b = m_a$ and $k'_b = k_a$; and*

III. $CT_{j_{k'_a}^{m'_a}} \geq CT_{j_{(k_a+1)}^{m_a}}$ *, where $m'_a = m_a$ and $k'_a = k_a + 1$; and*

IV. $CT_{j_{k'_b}^{m'_b}} \geq CT_{j_{(k_b+1)}^{m_b}}$ *, where $m'_b = m_b$ and $k'_b = k_b + 1$.*

Proof: Note that by swapping job $j_{k_a}^{m_a}$ with job $j_{k_b}^{m_b}$ on different machines, the k'^{th} positions of jobs $j_{k_a}^{m_a}$ and $j_{k_b}^{m_b}$ in the new schedule is k_b of machine m_b and k_a of machine m_a, respectively. By doing this, the completion time of all jobs which are assigned to machines m_a and m_b in the new schedule are either increased or not changed as follows:

- The $CT_{j_{l'}^{m'_a}}$ does not change, where $l' \in [1, k_a - 1]$ and $m'_a = m_a$
- The $CT_{j_{k'_b}^{m'_b}}$ increases or does not change, where $m'_b = m_a$ and $k'_b = k_a$
- The $CT_{j_{l'}^{m'_a}}$ is either increased or not changed, where $l' \in [k_a + 1, n_a]$ and $m'_a = m_a$
- The $CT_{j_{l'}^{m'_b}}$ does not change, where $l' \in [1, k_b - 1]$ and $m'_b = m_b$

- The $CT_{j_{k'_a}^{m'_a}}$ increases or does not change, where $m'_a = m_b$ and $k'_a = k_b$, and
- The $CT_{j_{l'}^{m'_b}}$ is either increased or not changed, where $l' \in [k_b + 1, n_b]$ and $m'_b = m_b$

The minimum increase in ΔZ satisfies when at least one job belonging to both $[k_a, n_a]$ of machine m_a and $[k_b, n_b]$ of machine m_b in the new schedule is increased; otherwise, the minimum increase in ΔZ is equal to zero.

Lemma 3.1 and 3.2 are illustrated in Figure 20. It represents exchanging job i and k on the same machine and also exchanging job j and f on different machines. The schedules are blocked after the backward inserted move of job k and j in the developed neighborhoods due to their release time. By assuming job i, j, and k belong to the s^{th} batch of group g, move $v_I(S_{sg}|1,3)$ related to machine M is stored in the ITL. If job f belongs to the s'^{th} batch of group p, move $v_E(M_{1_{g_{g'},g_p,g_{g''}}}|M_{2_{g_{p'},g_g,g_{p''}}})$ is stored in the CTL, in which job i is processed immediately before and after jobs of group p' and p'', respectively, and also job f is processed immediately before and after jobs of group g' and g'', respectively. In this case, $p' = p''$ and $g' = g''$. The first and second swapping moves are considered as sequencing and dividing move, respectively.

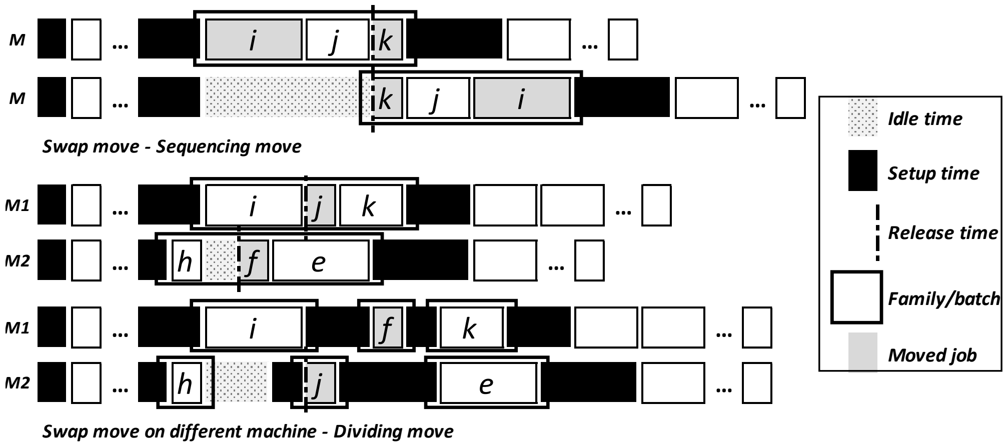


Figure 20. Illustration for lemmas 3-1 & 3-2

Suppose that each batch is considered as a whole job covering all jobs belonging to this batch, irrespective of how many jobs are included in the batch. Therefore, it assumes each batch is assigned to the k^{th} position of m^{th} machine in the initial schedule (s_k^m) and the k'^{th} position of m'^{th} machine in the new schedule ($s_{k'}^{m'}$), respectively, irrespective of which group the batch belongs to. Apart from this, $J_{js_k^m}$ and $J_{js_{k'}^{m'}}$ represent job j belonging to batch s assigned to the k^{th} position of m^{th} machine in the initial schedule and the k'^{th} position of m'^{th} machine in the new schedule, respectively. It is assumed that the position of all jobs belonging to the same batch s $(J_{js_k}{}^{m})$ is the same as the position of this batch on m^{th} machine (s_k^m), where j can at most be equal to the number of jobs belonging to batch s (i.e., $J_{js_k}{}^{m} = s_k^m$, where $j \in s$). The release time and the start time of the first job belonging to batch s are considered as the release time $(r_{s_k^m})$ and the start time $(ST_{s_k^m})$ of this batch. Likewise, the completion time of the last job belonging to batch s is considered as the completion time of this batch $(CT_{s_k^m})$.

By considering the above mentioned assumptions, the following lemmas 4-1, 4-2, 5, 6-1 and 6-2 applicable for batches can be proven, similar to lemmas 1-1, 1-2, 2, 3-1, and 3-2 applicable for jobs, simply by substituting s_k^m and $s_{k'}^{m'}$ in place of j_k^m and $j_{k'}^{m'}$, respectively. In other words, by considering just one job in each batch, lemmas 4-1, 4-2, 5, 6-1 and 6-2 are reduced to lemmas 1-1, 1-2, 2, 3-1, and 3-2, respectively. Therefore, considering more than one job for at least one batch guarantees to get the same conclusion. As a result, the completion time of a job belonging to a batch is not changed as long as the start time of this batch is not changed. On the other hand, the completion time of a job belonging to a batch, with its start time increased in the new schedule, is either increased or not changed.

- **Definition 3 (*Forward Move*):** By applying a forward move (insert or swap), batch s_k^m can be inserted into the k'^{th} position between batches on the same machine m, so that $k' > k$ $(k' \in [k+1, n])$.
- **Definition 4 (*Backward Move*):** By applying a backward move (insert or swap), batch s_k^m can be inserted into the k'^{th} position between batches on the same machine m, so that $k' < k$ $(k' \in [1, k-1])$.

Lemma 4-1: *Regardless of which batch of the same group or a different group batch s_{k-1}^m will be processed after, how many jobs each batch includes, due date and release time of other jobs, applying a forward inserting move for batch s_{k-1}^m on the same machine provides no improvement in the objective function value, if*

I. $ST_{s_{k'}^{m'}} \geq ST_{s_k^m}$ *, where $k' = k-1$ and $m = m'$; and*

II. $ST_{s_{(k'+1)'}^{m'}} \geq ST_{s_{(k')}^m}$ *, where $k' \in [k, n]$ and $m = m'$.*

Proof: Note that by inserting forwardly batch s_{k-1}^m at any available position on the same machine, the k'^{th} position of the batch s in the new schedule is greater than its $(k-1)^{th}$ position in the initial schedule $(k' >$

$k-1$ $(k' \in [k,n])$). By doing this, the completion time of all jobs belonging to all batches assigned to the same machine on the new schedule are either increased or not changed as follows:

- The $CT_{J_{js_{l'}^m}}$ does not change, where $j \in s_{l'}^m$ and $l' \in [1, k-2]$
- The $CT_{J_{js_{l'}^m}}$ is either increased or not changed, where $j \in s_{l'}^m$ and $l' \in [k-1, k'-1]$
- The $CT_{J_{js_{k'}^m}}$ increases, where $j \in s_{k'}^m$, and
- The $CT_{J_{js_{l'}^m}}$ is either increased or not changed, where $j \in s_{l'}^m$ and $l' \in [k'+1, n]$.

Therefore, the minimum increase in ΔZ due to the change in the completion time of job(s) j belonging to batch s inserted forwardly and its tardiness can be evaluated as $\Delta Z \geq \sum_{j\in[s_{k'}^m]} w_j \left(\alpha \left(CT_{J_{js_{k'}^m}} - CT_{J_{js_{k-1}^m}}\right) + \beta \left(T_{J_{js_{k'}^m}} - T_{J_{js_{k-1}^m}}\right)\right)$, where $T_{J_{js_{k'}^m}}$ and $T_{J_{js_{k-1}^m}}$ are the tardiness of job j belonging to batch s assigned to the k'^{th} and $(k-1)^{th}$ in the new and initial schedules, respectively. The tardiness of j_k^m is equal to $\max\{0, CT_{j_k^m} - d_j\}$. If the due date of job j belonging to batch s (d_{js}) is equal to or greater than its completion time in the new schedule $(d_{js} \geq CT_{J_{js_{k'}^m}})$, then $\Delta Z \geq \sum_{j\in[s_{k'}^m]} w_j \left(\alpha \left(CT_{J_{js_{k'}^m}} - CT_{J_{js_{k-1}^m}}\right)\right)$.

Lemma 4-2: *Regardless of which batch of the same group or a different group batch s_k^m will be processed after, how many jobs each batch includes, due date and release time of other jobs, applying a backward inserting move for s_k^m on the same machine provides no improvement in the objective function value, if*

I. $ST_{s_{k'}^{m'}} \geq ST_{s_k^m}$ *, where $k' \in [1, k-1]$ and $m = m'$; and*

II. $ST_{s_{k'}^{m'}} \geq ST_{s_{k+1}^m}$ *, where $k' = k+1$ and $m = m'$.*

Proof: Note that by inserting backwardly batch s_k^m at any available position on the same machine, the k'^{th} position of batch s in the new schedule is less than its k^{th} position in the initial schedule $(k' < k$ $(k' \in [1, k-1]))$. By doing this, the completion time of all jobs belonging to all batches assigned to the same machine in the new schedule are either increased or not changed as follows:

- The $CT_{J_{js_{l'}^m}}$ does not change, where $j \in s_{l'}^m$ and $l' \in [1, k'-1]$
- The $CT_{J_{js_{k'}^m}}$ is either increased or not changed, where $j \in s_{k'}^m$

- The $CT_{J_{js_{l'}^{m}}}$ is increased, where $j \in s_{l'}^{m}$ and $l' \in [k'+1, k]$ (common job(s)), and
- The $CT_{J_{js_{l'}^{m}}}$ is either increased or not changed, where $j \in s_{l'}^{m}$ and $l' \in [k+1, n]$.

Therefore, the minimum increase in ΔZ due to the change in the completion time of all common job(s) belonging to the batch(s) and its tardiness can be evaluated as $\Delta Z \geq \sum_{l \in [k', k-1], l' \in [k'+1, k]} w_j \left(\alpha\left(CT_{J_{js_{l'}^{m}}} - CT_{J_{js_{l}^{m}}}\right) + \beta\left(T_{J_{js_{l'}^{m}}} - T_{J_{js_{l}^{m}}}\right)\right)$ by considering one-to-one correspondence between l and l' (meaning k' with $k'+1$, and so on to finally consider $k-1$ with k), which includes the common jobs(s) belonging to the batch(s), which is processed after $J_{js_{k'}^{m}}$ (the last job j of batch s on k'^{th} position) in the new schedule and before $J_{js_{k+1}^{m}}$ (the first job j of batch s on $k+1^{th}$ position) in the initial schedule. If the due date of the common job(s) assigned to $\left[s_{k'+1}^{m}, s_{k}^{m}\right]$ is equal to or greater than its completion time on the new schedule, the difference in the objective function value can be written as $\Delta Z \geq \sum_{l \in [k', k-1], l' \in [k'+1, k]} w_j \left(\alpha\left(CT_{J_{js_{l'}^{m}}} - CT_{J_{js_{l}^{m}}}\right)\right)$.

Lemma 5: *Regardless of which batch of the same group or a different group batch $s_{k-1}^{m_a}$ will be processed after, how many jobs each batch includes, due date and release time of other jobs, inserting batch $s_{k-1}^{m_a}$ in any available position between batches on a different machine (m_b) provides no improvement in the objective function value, if*

I. $ST_{s_{k'}^{m'_a}} \geq ST_{s_{k}^{m_a}}$ *, where* $k' = k-1$ *and* $m'_a = m_a$ *; and*

II. $ST_{s_{(k'+1)'}^{m'_a}} \geq ST_{s_{(k')}^{m_a}}$ *, where* $k' \in [1, n_b]$ *and* $m'_a = m_a$ *; and*

III. $ST_{s_{k'}^{m'_a}} \geq ST_{s_{k-1}^{m_a}}$ *, where* $k' \in [1, n_b]$ *and* $m'_a = m_b$ *.*

Proof: Note that by inserting batch $s_{k-1}^{m_a}$ on a different machine m_b, the k'^{th} position of the batch s in the new schedule can be any available position as $k' \in [1, n_b]$. By doing this, the completion time of all jobs belonging to all batches assigned to machines m_a and m_b in the new schedule, are either increased or not changed as follows:

- The $CT_{J_{js_{l'}^{m'_a}}}$ does not change, where $j \in s_{l'}^{m'_a}$, $l' \in [1, k-2]$, and $m'_a = m_a$

- The $CT_{J_{js_{l'}^{m'a}}}$ is either increased or not changed, where $j \in s_{l'}^{m'_a}$, $l' \in [k-1, n_a - 1]$, and $m'_a = m_a$
- The $CT_{J_{js_{l'}^{m'b}}}$ does not change, where $j \in s_{l'}^{m'_b}$, $l' \in [1, k' - 1]$, and $m'_b = m_b$
- The $CT_{J_{js_{k'}^{m'a}}}$ increases or does not change, where $j \in s_{k'}^{m'_b}$, $k' \in [1, n_b]$, and $m'_a = m_b$, and
- The $CT_{J_{js_{l'}^{m'b}}}$ is either increased or not changed, where $j \in s_{l'}^{m'_b}$, $l' \in [k'+1, n_b + 1]$, and $m'_b = m_b$.

The minimum increase in ΔZ is satisfied when at least one job belonging to batches at both $[k-1, n_a - 1]$ interval of machine m_a and $[k', n_b + 1]$ interval of machine m_b in the new schedule is increased; otherwise, the minimum increase in ΔZ is equal to zero.

Lemma 6-1: *Regardless of which batches of the same group or different groups batches $s_{k_a}^m$ and $s_{k_b}^m$ will be swapped, how many jobs each batch includes, due date and release time of other jobs, swapping two batches $s_{k_a}^m$ and $s_{k_b}^m$ on the same machine, where $k_b > k_a$ provides no improvement in the objective function value, if*

I. $ST_{s_{k_b'}^{m'}} \geq ST_{s_{k_b}^m}$ *, where $k_b' = k_a$ and $m' = m$; and*

II. $ST_{s_{k_b'}^{m'}} \geq ST_{s_{k_b+1}^m}$ *, where $k_b' = k_b + 1$ and $m' = m$.*

Proof: Note that by swapping batch $s_{k_b}^m$ with batch $s_{k_a}^m$ processed before $s_{k_b}^m$ on the same machine, the k'^{th} positions of the batches $s_{k'_b}^{m'}$ and $s_{k'_a}^{m'}$ in the new schedule is k_a and k_b, respectively. By doing this, the completion time of all jobs belonging to all batches assigned to the machine m in the new schedule are either increased or not changed as follows:

- The $CT_{J_{js_{l'}^m}}$ does not change, where $j \in s_{l'}^m$ and $l' \in [1, k_a - 1]$
- The $CT_{J_{js_{k_b'}^m}}$ increases or does not change, where $j \in s_{k_b'}^m$
- The $CT_{J_{js_{l'}^m}}$ is increased, where $j \in s_{l'}^m$ and $l' \in [k_a + 1, k_b - 1]$
- The $CT_{J_{js_{k_a'}^m}}$ increases, where $j \in s_{k_a'}^m$, and

- The $CT_{J_{js_{l'}^m}}$ is either increased or not changed, where $j \in s_{l'}^m$ and $l' \in [k_b, n]$.

The minimum increase in ΔZ due to the change in the completion time of all jobs j belonging to batch s assigned to $[k_a + 1, k_b]$ in the new schedule and their tardiness can be evaluated as $\Delta Z \geq \sum_{l \in [k_a, k_b - 1], l' \in [k_a + 1, k_b]} w_j \left(\alpha \left(CT_{J_{js_{l'}^m}} - CT_{J_{js_l^m}}\right) + \beta \left(T_{J_{js_{l'}^m}} - T_{J_{js_l^m}}\right)\right)$, by considering one-to-one correspondence between l and l' (meaning k_a with $k_a + 1$, and so on to finally consider $k_b - 1$ with k_b). If the due date of the jobs belonging to batches assigned to $[s_{(k_b'+1)}^m, s_{k_a'}^m]$ is equal to or greater than their completion times on the new schedule, then $\Delta Z \geq \sum_{l \in [k_a, k_b - 1], l' \in [k_a + 1, k_b]} w_j \left(\alpha \left(CT_{J_{js_{l'}^m}} - CT_{J_{js_l^m}}\right)\right)$.

Lemma 6-2: *Regardless of which batches of the same group or different groups batches $s_{k_a}^{m_a}$ and $s_{k_b}^{m_b}$ will be swapped, how many jobs each batch includes, due date and release time of other jobs, swapping two batches $s_{k_a}^{m_a}$ and $s_{k_b}^{m_b}$ on different machines ($m_a \neq m_b$) provides no improvement in the objective function value, if*

I. $ST_{s_{k_a'}^{m_a'}} \geq ST_{s_{k_a}^{m_a}}$ *, where $m'_a = m_b$ and $k_a' = k_b$*

II. $ST_{s_{k_b'}^{m_b'}} \geq ST_{s_{k_b}^{m_b}}$ *, where $m'_b = m_a$ and $k_b' = k_a$*

III. $ST_{s_{k_a'}^{m_a'}} \geq ST_{s_{(k_a+1)}^{m_a}}$ *, where $m'_a = m_a$ and $k_a' = k_a + 1$, and*

IV. $ST_{s_{k_b'}^{m_b'}} \geq ST_{s_{(k_b+1)}^{m_b}}$ *, where $m'_b = m_b$ and $k_b' = k_b + 1$.*

Proof: Note that by swapping batch $s_{k_a}^{m_a}$ with batch $s_{k_b}^{m_b}$ on different machines, the k'^{th} positions of the batches $s_{k_a}^{m_a}$ and $s_{k_b}^{m_b}$ in the new schedule is k_b of machine m_b and k_a of machine m_a, respectively. By doing this, the completion time of all jobs belonging to all batches, which are assigned to the machines m_a and m_b in the new schedule, are either increased or not changed as follows:

- The $CT_{J_{js_{l'}^{m'a}}}$ does not change, where $j \in s_{l'}^{m_a'}$, $l' \in [1, k_a - 1]$, and $m'_a = m_a$
- The $CT_{J_{js_{k_b'}^{m'b}}}$ increases or does not change, where $j \in s_{k_b'}^{m_b'}$, $k'_b = k_a$, and $m'_b = m_a$

- The $CT_{J_{js_{l'}^{m'_a}}}$ is either increased or not changed, where $j \in s_{l'}^{m'_a}$, $l' \in [k_a + 1, n_a]$, and $m'_a = m_a$
- The $CT_{J_{js_{l'}^{m'_b}}}$ does not change, where $j \in s_{l'}^{m'_b}$, $l' \in [1, k_b - 1]$, and $m'_b = m_b$
- The $CT_{J_{js_{k'_a}^{m'_a}}}$ increases or does not change, where $j \in s_{k'_a}^{m'_a}$, $k'_a = k_b$, $m'_a = m_b$, and
- The $CT_{J_{js_{l'}^{m'_b}}}$ is either increased or not changed, where $j \in s_{l'}^{m'_b}$, $l' \in [k_b + 1, n_b]$ and $m'_b = m_b$

The minimum increase in ΔZ is satisfied when at least one job belonging to both $[k_a, n_a]$ of machine m_a and $[k_b, n_b]$ of machine m_b in the new schedule is increased; otherwise, the minimum increase in ΔZ is equal to zero.

There are several important points related to developed lemmas.

- First, in all of the above cases related to lemmas, the schedule(s) is blocked due to either release time of a job(s)/batch(es) or the new setup time for a moved job(s)/batch(es), which can both freeze some jobs/batches for backward movement on a machine(s), resulting in no improvement.
- Second, an ineffective neighbor move is determined for a particular stage, irrespective of the other stages.
- Third, the completion time of a job(s) related to immediately prior stage is considered as its release time(s) for current stage in order to identify an ineffective move with the help of lemmas. This prior stage is where the job had its latest operation (considering stage skipping).
- Fourth, when the outcome of a move makes no contribution to improve the objective function value (ΔZ) in a particular stage, the completion time of jobs for this stage and, subsequently, the release time of these jobs for the immediately following stage are either increased or not changed. This scenario repeats for each of two consecutive stages, up to the last stage.

The findings from computational tests in a particular stage of HFS are summarized in Table 12. Applying lemmas can significantly reduce computational times by about 24.5%, on average (up to 40%). It is worth noting that this reduction will be more significant when lemmas are implemented in all stages. Also, the reduction in computational times is far more significant if the lemmas are implemented in solving medium and large size problems than small ones.

Table 12. Performance of the TS-based algorithm with and without the lemmas

Ex. Problem	# of groups	# of machines	# of all jobs	Without lemmas		With Lemmas		Percentage of Improvement
				Obj Func Val.	Comp Time (Sec)	Obj Func Val.	Comp Time (Sec)	
1	7	5	22	1228.4	2305.0	1228.4	2212.8	4.0%
2	7	4	19	1806.6	1061.1	1806.6	1050.5	1.0%
3	7	5	24	4544.4	2566.5	4544.4	2284.2	11.0%
4	4	4	17	2322.6	853.2	2322.6	767.9	10.0%
5	5	5	17	2468.8	118.8	2468.8	115.2	3.0%
6	5	3	23	3230.5	364.2	3230.5	327.8	10.0%
7	8	4	25	2501.0	8767.0	2501.0	7539.6	14.0%
8	8	3	24	3551.5	3549.6	3551.5	2484.7	30.0%
9	7	3	26	6245.5	9878.7	6245.5	7606.6	23.0%
10	8	4	28	4695.2	5905.5	4695.2	4488.2	24.0%
11	7	3	26	3255.4	2921.1	3255.4	1898.7	35.0%
12	6	3	24	3667.0	1618.2	3667.0	1375.5	15.0%
13	9	3	29	3989.6	24034.1	3989.6	16583.5	31.0%
14	9	3	30	4041.6	17044.7	4041.6	13635.8	20.0%
15	13	4	39	3489.6	29510.1	3489.6	17706.1	40.0%
16	9	3	27	3621.0	5369.4	3621.0	4188.1	22.0%
17	11	3	33	5939.8	11319.7	5939.8	8376.5	26.0%
18	12	3	36	5591.0	37176.0	5591.0	24536.2	34.0%

4.2.9. Calibration of the meta-heuristic algorithms

Based on preliminary runs, all algorithms are allowed to run over a 2-h time limit considered to be the point at which the algorithms become mature. With respect to the stage-based interdependency strategy, the corresponding parameters are tuned/calculated separately in each stage, based on the performance of the algorithms in the above-mentioned time interval.

The TS parameters are tuned by performing multiple regressions on some parameters of the problem, i.e., number of machines (M), number of batches (B), and average number of jobs in each batch ($\bar{J}$). The tuned parameters for the TS/PR algorithm, shown in Table 13, include the tabu list size (TLS), the index list size (ILS), and the maximum iterations without improvement (MIWOI), for all search levels of TS-based algorithms, as well as P_{size} for all PR procedures. All parameters are tuned for each problem structure determined in Section 5. Since B and/or $\bar{J}$ might have different values in each level/stage of TS, these parameters should be tuned whenever the TS-based algorithm moves across levels or stages. The empirical formulae are obtained for TS parameters with the help of DATAFIT (1995). During TS/PR, two types of TS are implemented for candidate solutions: slight TS including only the CTS and strong TS including the CTS, OTS, and ITS. The TS parameters of slight TS follow the central TLS (CTLS), central ILS (CILS), and central MIWOI (MCIWOI) of strong TS. P_{size} is dependent on the applied path construction technique and level of the problem.

Table 13. Empirical formulae for TS parameters

	(Small, Small)	*(Small, Large)*	*(Large, Small)*	*(Large, Large)*
CTLS	2	2	$\lceil 0.242 + 0.52B - 0.48M + 0.1\bar{J} \rceil$	$\lceil 2.5 + 0.05B + 0.06M - 0.38\bar{J} \rceil$
CILS	$\lceil 0.82 + 0.74B - 0.7M + 0.5\bar{J} \rceil$	$\lceil 2.455 + 0.42B - 0.15M - 0.05\bar{J} \rceil$	$\lceil -5.5 + 3.545B - 1.05M - 1.5\bar{J} \rceil$	$\lceil 12.25 + 1.62B + 0.54M - 2.48\bar{J} \rceil$
MCIWOI	$\lceil 0.35 + 0.48B - 0.3M + 0.4\bar{J} \rceil$	$\lceil 1.05 + 0.28B - 0.2M + 0.1\bar{J} \rceil$	$\lceil -8.283 + 1.09B - 1.11M + 2.6\bar{J} \rceil$	$\lceil -28.48 + 3.2B + 1.55M - 0.05\bar{J} \rceil$
OTLS	$\lceil -0.7B + 1.35M + 2.75\bar{J} \rceil$	$\lceil 0.44B - 0.68M + 1.55\bar{J} \rceil$	$\lceil -0.6B + 1.8M + 1.25\bar{J} \rceil$	$\lceil 0.38B + 0.1M - 0.09\bar{J} \rceil$
OILS	$\lceil 1.21B - 0.64M + 0.59\bar{J} \rceil$	$\lceil 0.384B + 0.08M + 0.13\bar{J} \rceil$	$\lceil 1.5B - 1.1M - 1.123\bar{J} \rceil$	$\lceil 0.28B + 3.39M - 1.38\bar{J} \rceil$
MOIWOI	$\lceil 1.82B - 0.35M + 1.66\bar{J} \rceil$	$\lceil 0.456B - 0.05M \rceil$	$\lceil 0.473B - 0.6M - 0.09\bar{J} \rceil$	$\lceil 0.05B + 1.01M - 0.23\bar{J} \rceil$
ITLS	$\lceil 0.88B - 0.99M + 1.342\bar{J} \rceil$	$\lceil 0.29B - 0.31M + 0.98\bar{J} \rceil$	$\lceil 0.05B + 0.2M + 0.65\bar{J} \rceil$	$\lceil -0.06B + 1.1M + 0.152\bar{J} \rceil$
IILS	2	2	2	$\lceil 0.03B + 0.345M + 0.212\bar{J} \rceil$
MIIWOI	1	1	1	$\lceil 0.339M - 0.03\bar{J} \rceil$
$\boldsymbol{P_{size}}$	10	15	15	20
LCS	10	8	7	5
LCS&Block	8	7	5	4
LCS&Swap	8	6	4	3

LCS, LCS&Block, and LCS&Swap stand for LCS-based, LCS- and block-based, LCS- and swap-based construction techniques, respectively. Ceiling brackets $\lceil . \rceil$ rounds a number to its larger integer.

The PSO parameters are calculated by experimental design techniques. Based on extensive experiments using test problems different from those implemented for TS/PR, the PSO parameters are tuned according to the values of Table 14.

Table 14. The PSO algorithm parameters

Swarm Size	*Lower & upper bounds*
ω^{min} & ω^{max}	0.4 & 2.0
${c_1}^{min}$ & ${c_1}^{max}$	0.4 & 2.4
${c_2}^{min}$ & ${c_2}^{max}$	0.4 & 2.4
vel^{min} & vel^{max}	-4.0 & 4.0
x^{min} & x^{max}	0.0 & 4.0
a	10
Itr_{max}	10000

During the evolution process of PSO, the non-linear dynamic coefficients are considered at each stage as follows:

$$\omega = \omega^{min} + \left(\left(\omega^{max} - \omega^{min}\right) \Big/ \left(1 + \exp\left(\left(-a(Itr_{max} - t)\right) \Big/ Itr_{max} \right)\right) \right) \tag{4.72}$$

$$c_1 = {c_1}^{min} + \left(\left({c_1}^{max} - {c_1}^{min}\right) \Big/ \left(1 + \exp\left(\left(-a(Itr_{max} - t)\right) \big/ Itr_{max}\right)\right) \right) \tag{4.73}$$

$$c_2 = {c_2}^{min} + \left(\left({c_2}^{max} - {c_2}^{min}\right) \Big/ \left(1 + \exp\left(-at \big/ Itr_{max}\right)\right) \right) \tag{4.74}$$

$\omega^{max}/{c_1}^{max}/{c_2}^{max}$ represents the highest value of $\omega/c_1/c_2$ and $\omega^{min}/{c_1}^{min}/{c_2}^{min}$ represents the lowest value of $\omega/c_1/c_2$. Parameter a has a constant value. r_1 and r_2 are independently generated by $unif = [0, 1]$. The appropriate value of χ in the case of meeting condition $c_1 + c_2 > 4$ is determined by $\chi = 2/(C - 2 + \sqrt{C^2 - 4C})$, where $C = c_1 + c_2$; otherwise $\chi = 1$. The tuned parameters for the number of iterations for LSA (Itr_{LSA}) and P'_{size} are shown in Table 15.

Table 15. Empirical formulae for PSO/LSA parameters

	(Small, Small)	*(Small, Large)*	*(Large, Small)*	*(Large, Large)*
$\boldsymbol{Itr_{LSA}}$	$\left\lceil 0.2\times\sum_{i\in I^k} J_i^k \right\rceil$	$\left\lceil 0.16\times\sum_{i\in I^k} J_i^k \right\rceil$	$\left\lceil 0.14\times\sum_{i\in I^k} J_i^k \right\rceil$	$\left\lceil 0.1\times\sum_{i\in I^k} J_i^k \right\rceil$
$\boldsymbol{P'_{size}}$	$2\sum_{i\in I^k} J_i^k$	$2.45\sum_{i\in I^k} J_i^k$	$2.55\sum_{i\in I^k} J_i^k$	$3\sum_{i\in I^k} J_i^k$

5. LOWER BOUNDS

Due to the inherent complexity of combinatorial optimization problems such as the one considered in this research, the enormous number of constraints and variables introduced into the model typically results in a very large solution space. Since the research problem is strongly NP-hard (Section 4), it is not possible to optimally solve it within a reasonable computational time. Thus, the need for solving the problem in the hope of finding at least a solution for large industry-size problems has encouraged researchers to develop advanced heuristic and meta-heuristic algorithms. Since these non-exact algorithms do not guarantee solving a problem optimally, a mechanism should be established to evaluate the performance of theses algorithms.

In dealing with NP-hard problems, benchmarking, which is one of the most common approaches for this evaluation, is not a successful tool for comparison, because the best practices are also non-exact algorithms, perhaps producing highly inferior solutions. Apart from this, there is no benchmark in the literature that is related to the research problem addressed in this research. Therefore, developing a lower bound for the optimal solution as the baseline for the minimization problems can determine the relative performance of the non-exact algorithms. Since the optimal solution must be between the lower bound and the best solution obtained by a non-exact algorithm, the gap between this bound and the proposed algorithm can be considered as an indicator of the performance of that algorithm (Bozorgirad and Logendran 2014). The mentioned gap is composed of two intervals, i.e. the deviation of the proposed algorithm from the optimal solution, and the deviation of the lower bound from the optimal solution. Therefore, the smaller the gap, the higher the performance of the proposed algorithm. Also, the closer the lower bound is to the optimal solution, the more precise is the assessment of the proposed algorithm. Thus, in order to develop a tight lower bound, the second interval of this gap must be minimized.

5.1. Lower bounding mechanisms

There are several different techniques to develop a lower bound including LP relaxation, Benders decomposition, column generation, etc. The simplest lower bound for any IP problem is the linear programming (LP) relaxation of that problem, which is obtained by relaxing integrality constraints of the problem. Although the LP relaxation technique is very fast in finding a lower bound, the developed lower bound can have a large deviation from the optimal solution and thus very inferior compared to other techniques. Selective LP relaxation and iterative selective LP relaxation are extended version of LP relaxation in order to improve the quality of the developed lower bound (Mehravaran 2013).

Classical Benders decomposition enumerates values of certain variables for solving the problem, which is decomposed into a master problem (MP) and a set of sub-problems (SPs). For each set of enumerations, the values of certain variables are fixed and fed into the SPs. Solution of the SPs generates a Benders cut and must be satisfied in all subsequent solutions enumerated. Logic-Based Benders Decomposition (LBBD) was introduced by Hooker and Yan (1995) in the context of logic circuit verification. The LBBD is a manual decomposition technique that generalizes classical Benders decomposition technique (Hooker and Ottosson 2003).

The lower bounds obtained by LP relaxation techniques usually have a large deviation from the optimal solution. Benders decomposition, known as row generation, is more useful, when it is applied for a problem with a large number of constraints. In order to find tight lower bounds for large-size linear programming problems with a large number of variables, an efficient technique called *Column Generation* (CG) was developed. Initially starting from a model to cover a manageable part of the solution space, column generation discovers a feasible solution satisfying the problem and adds it to the latest partial model updated so far. Gradually, the model gets larger in size until it achieves a satisfactory solution to the entire problem. The columns are the solutions that are iteratively identified and added to the partial model. Typically, there are a large number of columns for large-size LP problems. Initially, working with all of these columns is practically impossible and it is equivalent to solving the associated mathematical programing model formulated for the problem in its entirety. Accordingly, many of the columns are initially left out, yet many of them may not make a contribution in identifying the optimal solution. The main idea behind this technique is to deal only implicitly with the columns of an optimization problem. This capability makes column generation one of the most prominent techniques in solving LPs with a huge number of variables.

5.2. Review of the literature related to column generation technique

Column generation is implemented to solve the IP models obtained by a Dantzig-Wolfe decomposition (Dantzig and Wolfe 1960), and was first implemented by Gilmore and Gomory (1961, 1963), as part of a heuristic algorithm to deal with the cutting stock problem. The embedding of column generation in a branch-and-bound (B&B) framework for solving a vehicle routing problem under time window constraints (Desrosiers et al. 1984) was a key step in solving large-size IPs to optimality. Wilhelm (2001) provided a technical review of column generation in integer programming and proposed three different types of column generation identified in the literature. Each of these types includes a master problem (MP) with restricted columns (RMP) that has to be optimized.

- Type I column generation uses an auxiliary model (AM) to generate an attractive set of columns, and then the RMP is optimized over those explicitly identified columns. In this type of column generation, there is no interaction between the RMP and AM.

- Type II column generation uses a price-out problem (POP) that interacts with the RMP to identify a non-basic column with the most negative reduced cost.

- Type III column generation applies Dantzig-Wolfe Decomposition (DWD) to the linear relaxation of an IP. The dual variables of the RMP comprise the objective function coefficients of one or more SPs, and each SP is solved to introduce improving columns into the RMP.

Using column generation to find lower bounds is not new in the literature of scheduling problems. Column generation has been used for many scheduling problems including parallel machines and flow shop. Van Den Akker et al. (1999) applied column generation technique for minimizing the total weighted completion time of n jobs on m identical-parallel machines. The problem was formulated as a set-covering problem with huge number of variables, n covering constraints, and a single side constraint. The numerical experiments revealed that the lower bound was very tight, and the linear program often resulted in integer solution. Thereafter, Van Den Akker et al. (2000) applied column generation technique to optimize a time-indexed formulation of a single-machine scheduling problem. The main disadvantage of their models was related to size. This difficulty is alleviated with the help of Dantzig-Wolfe decomposition.

Chen and Powell (1999a) used column generation for the problem of scheduling n jobs on m identical, uniform or unrelated-parallel machines with respect to minimizing a linear combination of total weighted completion time and weighted number of tardy jobs. Also, Chen and Powell (1999b) minimized the total weighted tardiness and earliness of n jobs with a non-restrictively large common due date on m identical-parallel machines with the help of this technique. Each column in their formulation represented a partial schedule, which was generated by single machine scheduling sub-problem. In both research, they formulated the problem as an integer program, and then reformulated the problem as a set partitioning problem. A B&B algorithm is implemented in order to find an optimal integer solution for the second problem. Later, Chen and Powell (2003) applied column generation to minimize the total weighted completion time of multiple job families including n jobs on m identical-parallel machines, with sequence-dependent or sequence-independent setup times. Chen and Lee (2002) proposed a similar approach developed by Chen and Powell (1999b) to solve problems with 40 jobs and 6 parallel machines within a reasonable computational time. They addressed the problem of scheduling n independent jobs on m identical-parallel machines to minimize the total earliness-tardiness penalty of the jobs. Lopes and de Carvalho (2007) used the column generation in each branch of a B&B algorithm, known as branch-and price (B&P) optimization algorithm, for the problem of scheduling a set of independent jobs, with release dates and due dates, on a set of unrelated-parallel machines with availability dates and sequence-dependent setup times, to minimize the total weighted tardiness. They used pseudo-polynomial algorithms based on dynamic programming to solve the SPs, together with an accelerating mechanism for column generation called primal box and a specific branching variable selection rule that significantly reduced the computational time.

Column generation has also been successfully implemented for flow shop scheduling problems. Bozorgirad and Logendran (2014) developed tight lower bounds for group scheduling in hybrid flow shop environment with respect to minimizing a linear combination of total weighted completion time and total weighted tardiness of all jobs assigned to pre-determined groups. Bülbül et al. (2004) proposed heuristic algorithms based upon column generation in order to minimize tardiness, earliness and work-in-process costs for a flow shop scheduling problem. Gelogullari and Logendran (2010) and Salmasi et al. (2010) implemented column generation for group scheduling problems in flow shop environment in the framework of a B&B algorithm in order to derive tight lower bounds. Then, they evaluated the performance of their meta-heuristics with the help of developed lower bounds.

5.3. Branch-and-Price algorithm

A B&P algorithm is developed to obtain a tight lower bound with the help of applying the column generation in each node, for the HFS scheduling problem addressed in this research. A decomposition of an MILP model is then presented, which is followed by branching rules for the B&P algorithm. Three types of decomposition of problems are developed in terms of three different MILP models, i.e., the MILP1, RMILP, and MILP3 models. Since there is not too much difference between the MILP1 and MILP2, the decomposition of problems is developed for only one of them. i.e., MILP1. Before going into the details of different types of decomposition of problems, a common decomposition technique is explained. Then, an acceleration technique and early termination of column generation are explained.

5.3.1. Dantzig-Wolfe Decomposition

Decomposing the problem into smaller problems which are computationally easier to solve is one of the successful techniques in dealing with large-size complex problems. Column generation is one of the decomposing techniques and divides the constraints of the MILP model into two parts, forming two individual yet independent problems: an MP composed of the objective function and the most difficult constraints that are dependent on each other; one or more SPs composed of the independent constraints forming block diagonal structure. In other words, the MP is a set of columns that defines the solution space, while columns are developed by the SPs. Any solution to the MILP model should satisfy both the MP and SPs. Each column is a vector that contains a cost coefficient as well as all constraint coefficients for one of the decision variables. Even though a problem has a restricted number of variables, it can have unlimited number of columns. Therefore, a restricted master problem (RMP) with limited number of columns (a subset of whole columns) is implemented in practice (Bozorgirad and Logendran 2014).

The column generation, in this research, applies Dantzig-Wolfe decomposition for the MILP model to decompose the problem (Bazaraa et al. 2011, Dantzig and Wolfe 1960, Wilhelm 2001). The dual variables of the RMP comprise the objective function coefficients of one or more SPs, and each SP is solved to introduce improving columns into the RMP. The optimal solution in the SP or the solution with the most negative reduced cost is transferred to the RMP and again the same process is repeated until no more solution with negative reduced cost can be identified in any of the SPs. At this time, the problem is solved to optimality (Barnhart et al. 1998, Lübbecke and Desrosiers 2005, Vanderbeck 2000, Vanderbeck and Wolsey 1996, Wilhelm 2001). Identifying a set of constraints that links all the other constraints together (linking constraints) is the basis of Dantzig-Wolfe decomposition. By relaxing linking constraints, the problem can be decomposed into a number of independent SPs.

5.3.1.1. DWD1

The MILP1 model moves back and forth between batching and scheduling phases. For all of the stages, the batch assignments to machines, batch sequences on machines, and job sequence within batches are determined in the scheduling phase, while the batch compositions of groups are determined in the batching phase. Since the optimal batch assignment to machines, batch sequence on machines, and job sequences within batches should be determined in the scheduling phase for each combination between batch compositions of groups related to all stages, which is developed in the batching phase, the solution space exponentially increases and, subsequently, this leads to unaffordable and unreasonable computational times.

In the MILP1 model, constraint (4.13) is the only linking constraint. With respect to this constraint, completion time of a job in each stage is coupled with the completion time of that job in the preceding stage. This preceding stage is where the job had its latest operation. By relaxing constraint (4.13), all the other constraints only include the decision variables corresponding to a particular stage at a time and, consequently, there is one SP for each stage, determining the partial batch sequences on machine(s) as well as job sequences within batches on that stage. These partial sequences are linked together with the help of constraint (4.13) in the RMP. In addition to the notations used in section 4.1.1, the following notations are defined to formulate the RMP and SPs.

$\mathcal{T}$	Set of columns, indexed by t	$\mathcal{T} = \{1,2,\dots,T\}$
T	Total number of columns, where each column represents the partial schedules for all the stages,	
X_{isj}^{kt}	The completion time of job j assigned to batch s of group i in stage k in column t	
T_{ij}^{t}	The tardiness of job j of group i in column t	
$\emptyset_{isj}^{kt}$	1 if job j is assigned to batch s of group i in stage k in column t; 0 otherwise	
Z_{ish}^{kt}	1 if batch s of group i is assigned to machine h in stage k in column t; 0 otherwise	
λ_t	1 if column t is selected; 0 otherwise	

Restricted linear master problem (RLMP)

$$Min\ Z = \sum\nolimits_{t\in\mathcal{T}} \left(\sum\nolimits_{i\in G}\sum\nolimits_{j\in g_i} w_{ij}\left(\sum\nolimits_{s\in g_i} \alpha . X_{isj}^{\left(st_{ij(m_{ij}+1)}\right)t} + \beta . T_{ij}^{t}\right)\right)\lambda_t \qquad (5.1)$$

Subject to:

$$\sum_{t\in\mathcal{T}}\left(X_{isj}^{st_{ij(r)}t}-X_{is'j}^{st_{ij(r-1)}t}-\sum_{h\in V_{ij}^{st_{ij(r)}}} t_{ijh}^{st_{ij(r)}}\left(Z_{ish}^{st_{ij(r)}t}\right)\right.$$
$$\left.-\left(\phi_{isj}^{st_{ij(r)}t}+\phi_{is'j}^{st_{ij(r-1)}t}-2\right)M\right)\lambda_t\geq 0; \qquad (5.2)$$

$$i=1,2,\dots,g;\ j=1,2,\dots,n_i;\ s=1,2,\dots,S_i^{st_{ij(r)}};s^{'}=1,2,\dots,S_i^{st_{ij(r-1)}};$$
$$r=1,2,\dots,m_{ij},m_{ij}+1; M\text{: }large\ number$$

$$\sum_{t\in\mathcal{T}}\lambda_t=1; \qquad (5.3)$$

$$\lambda_t\in\{0,1\}; \quad t\in\mathcal{T} \qquad (5.4)$$

This model is initiated with a given feasible solution as the first column. With the help of developed initial solution finding mechanism (section 4.2.2), the column generation algorithm is initialized. As the column generation progresses, the model iteratively includes more columns. Since all variables $X_{isj}^{st_{ij(r)}t}$, $Z_{ish}^{st_{ij(r)}t}$, $\phi_{isj}^{st_{ij(r)}t}$, and T_{ij}^{t} related to the original MILP model are assumed to be known, the only decision variable in the model is binary variables λ_t. The objective function of the RMP (5.1) minimizes the objective function value of the original MILP model over all columns that are introduced into the RMP. The linking constraint (5.2) relates the completion time of a job belonging to a batch of a group in each stage to completion time of that job in the prior stage, in which this job had an operation. Convexity constraint (5.3) ensures that a convex combination of columns is selected in each iteration of the algorithm. And finally, the integrality of the problem is guaranteed by the binary constraint (5.4).

The RMP does not include all the possible columns and is restricted in the number of columns. Instead, all feasible columns are generated with the help of the SPs, and the best one will be introduced into the RMP. The column contributed to the most improvement to the objective function of the RMP is known as the best column, which is the column with the minimum reduced cost. Therefore, the objective function of the SPs determine the column with the minimum reduced cost. If this column has a negative reduced cost, it will be introduced into the RMP. This process will be repeated between the RMP and SPs until no further improvement to the RMP is obtained. This is considered as the stopping criteria for the column generation algorithm. In final stage, the optimal solution of this RMP will be equivalent to the optimal solution for the unrestricted master problem. From a linear programming point of view, the reduced cost of the RMP can be identified based upon the dual of this problem. Therefore, the integrality constraints (5.4) are relaxed ($\lambda_t \geq 0$) and then the dual of this problem is developed as follows:

$\Phi_{ijs\acute{s}r}$	Dual variable associated with constraint (5.2)
ψ	Dual variable associated with constraint (5.3)

Dual of the linear master problem (DLMP)

$$Max\ Z = \psi \tag{5.5}$$

Subject to:

$$\begin{aligned}
&\sum_{i\in G}\sum_{j\in g_i}\sum_{s\in g_i}\sum_{s'\in g_i}\sum_{r=1}^{m_{ij}+1}\Bigg(X_{isj}^{st_{ij(r)}t} - X_{is'j}^{st_{ij(r-1)}t} - \sum_{h\in V_{ij}^{st_{ij(r)}}} t_{ijh}^{st_{ij(r)}}\left(Z_{ish}^{st_{ij(r)}t}\right)\\
&\qquad - \left(\phi_{isj}^{st_{ij(r)}t} + \phi_{is'j}^{st_{ij(r-1)}t} - 2\right)M\Bigg)\times\Phi_{ijs\acute{s}r} + \psi\\
&\qquad \leq \sum_{i\in G}\sum_{j\in g_i} w_{ij}\left(\sum_{s\in g_i}\alpha.X_{isj}^{\left(st_{ij\left(m_{ij}+1\right)}\right)t} + \beta.T_{ij}^{t}\right);\\
&\qquad t \in \mathcal{T};
\end{aligned} \tag{5.6}$$

$$\begin{aligned}
&\Phi_{ijs\acute{s}r} \geq 0;\\
&\forall i = 1,2,\dots,g; j = 1,2,\dots,n_i; s = 1,2,\dots,g_i^{st_{ij(r)}}; s' = 1,2,\dots,g_i^{st_{ij(r-1)}}; r = 1,2,\dots,m_{ij};
\end{aligned} \tag{5.7}$$

$$\psi \quad unrestricted; \tag{5.8}$$

As mentioned earlier, each column in the RMP includes all stages in the hybrid flow shop. Apart from this, there is only one constraint (5.6) for this column in the DLMP. Therefore, there should be only one unified SP, which includes all stages simultaneously. Since all decision variables corresponding to each stage in this unified SP are totally independent of each other, the SP is decomposed into multiple SPs related to each stage. In other words, each SP assigned to each stage includes all variables corresponding to a particular stage. As a result, an exhaustive combination enumeration between batch compositions of all groups in all stages is excluded, but an enormous number of combinations between batch compositions of all groups in each stage still remains. In developing the SPs, two extra virtual stages are introduced as follows:

- Stage (0) is an initial virtual stage, where all jobs begin their process from. Run time of all jobs in this stage is equal to zero, and the only constraint in this stage is constraint (4.9), which defines the non-zero job release times. Therefore, the completion time of each job is equal to its release time in initial virtual stage.
- Stage ($m + 1$) is a final virtual stage, where all jobs will be completed. Run time of all jobs in this stage is zero, and the only constraint sets in this stage are constraints (4.14) and (4.15), which are the tardiness and sign constraints, respectively.

With the help of these two virtual stages, the assumptions of non-zero job release time as well as bi-criteria objective function are excluded from middle stages ($k \in K$). Therefore, the objective function associated with middle stages of SPs is related to only one criterion, i.e., total completion time. These SPs are presented in the following based upon different stages:

Sub-problems (SPs)

$\boldsymbol{SP(0), st_{ij(0)} = 0, \forall\, i \in I^0 \;\&\; j \in J_i^0}$

$$Min\ Z_{sp(1)} = \sum_{i \in G} \sum_{j \in g_i} \sum_{s' \in g_i} \left(\left(\sum_{s \in g_i} \Phi_{ijs\acute{s}1} \right) X_{is'j}^{0t} \right) + \sum_{i \in G} \sum_{j \in g_i} \sum_{s' \in g_i} \left(\left(\sum_{s \in g_i} \Phi_{ijs\acute{s}1} \right) \phi_{is'j}^{0t} \right) \tag{5.9}$$

Subject to:

$$X_{isj}^{0} \geq r_{ij};\ i \in I^0, j \in J_i^0. \tag{5.10}$$

$$\boldsymbol{SP(k), 1 \leq k \leq m, for\ st_{ij(r)} = k, \forall i = 1, 2, \ldots, g \;\&\; j = 1, 2, \ldots, n_i \;\&\; r = 1, 2, \ldots, m_{ij}} \tag{5.11}$$

$$\begin{aligned} Min\ Z_{sp(k)} = & \sum_{i\in G}\sum_{j\in g_i}\sum_{s'\in g_i}\left(\left(\sum_{s\in g_i}\Phi_{ijs\acute{s}(r+1)}\right)X_{is'j}^{kt}\right) \\ & + \sum_{i\in G}\sum_{j\in g_i}\sum_{s\in g_i}\left(\left(\sum_{s'\in g_i}\Phi_{ijs\acute{s}r}\right)X_{isj}^{kt}\right) \\ & + \sum_{i\in G}\sum_{j\in g_i}\sum_{s'\in g_i}\left(\left(\sum_{s\in g_i}\Phi_{ijs\acute{s}r}\right)M\times\phi_{isj}^{kt}\right) \\ & + \sum_{i\in G}\sum_{j\in g_i}\sum_{s\in g_i}\left(\left(\sum_{s'\in g_i}\Phi_{ijs\acute{s}(r+1)}\right)M\times\phi_{i\acute{s}j}^{kt}\right) \\ & + \sum_{i\in G}\sum_{j\in g_i}\sum_{s\in g_i}\left(\left(\sum_{h\in V_{ij}^k}\left(\sum_{s'\in g_i}\Phi_{ijs\acute{s}r}\right)t_{ijh}^k\right)Z_{ish}^{kt}\right) \\ & + 2\left(\sum_{i\in G}\sum_{j\in g_i}\sum_{s\in g_i}\sum_{s'\in g_i}\Phi_{ijs\acute{s}r}\right)\times M \end{aligned}$$

Subject to:

$$\sum_{s\in S_i^k}\emptyset_{isj}^k = 1 \qquad (5.12)$$

$$i \in I^k;\ j \in J_i^k;\ k \in K;$$

$$\sum_{h\in V^k} Z_{ish}^k \le 1 \qquad (5.13)$$

$$i \in I^k;\ s \in S_i^k;\ k \in K;$$

$$\sum_{h\in V^k} Z_{ish}^k \ge \emptyset_{isj}^k \qquad (5.14)$$

$$i \in I^k;\ j \in J_i^k;\ s \in S_i^k;\ k \in K;$$

$$\sum_{j\in J_i^k}\emptyset_{isj}^k \ge \sum_{h\in V^k}\left(LB_{ih}^k\right)Z_{ish}^k \qquad (5.15)$$

$$s \in S_i^k;\ i \in I^k;\ k \in K;$$

$$X_{isj}^k + M\left(1 - A_{ptis}^k\right) + M\left(1 - Z_{ish}^k\right) + M\left(1 - Z_{pth}^k\right) + M\left(1 - \emptyset_{isj}^k\right) \ge C_{pt}^k + S_{pih}^k + t_{ijh}^k \qquad (5.16)$$

$$i, p \in I^k (p \le i; p = i \rightarrow t < s);\ j \in J_i^k;\ h \in v_{ij}^k\ ;\ k \in K;\ s \in S_i^k;\ t \in g_p^k;\ M\!: large\ number;$$

$$X_{ptj}^{k} + M\left(A_{ptis}^{k}\right) + M\left(1 - Z_{ish}^{k}\right) + M\left(1 - Z_{pth}^{k}\right) + M\left(1 - \emptyset_{ptj}^{k}\right) \geq C_{is}^{k} + S_{iph}^{k} + t_{pjh}^{k}$$

$$i, p \in I^{k} (p \leq i; p = i \rightarrow t < s);\ j \in J_{p}^{k};\ h \in v_{pt}^{k};\ k \in K;\ s \in S_{i}^{k};\ t \in g_{p}^{k};\ M: large\ number; \quad (5.17)$$

$$X_{isj}^{k} + M\left(1 - \emptyset_{isj}^{k}\right) \geq \sum\nolimits_{h \in v_{ij}^{k}} \left(a_{h}^{k} + S_{0ih}^{k} + t_{ijh}^{k}\right) Z_{ish}^{k}$$

$$i \in I^{k};\ j \in J_{i}^{k}; s \in S_{i}^{k};\ k \in K; \quad (5.18)$$

$$X_{isj}^{k} - X_{isq}^{k} + M\left(Y_{isjq}^{k}\right) + M\left(1 - \emptyset_{isj}^{k}\right) + M\left(1 - \emptyset_{isq}^{k}\right) \geq \sum\nolimits_{h \in v_{ij}^{k} \cap v_{iq}^{k}} t_{ijh}^{k}\left(Z_{ish}^{k}\right)$$

$$i \in I^{k};\ j, q \in J_{i}^{k} (j < q); k \in K; s \in S_{i}^{k};\ M: large\ number; \quad (5.19)$$

$$X_{isq}^{k} - X_{isj}^{k} + M\left(1 - Y_{isjq}^{k}\right) + M\left(1 - \emptyset_{isj}^{k}\right) + M(1 - \emptyset_{isq}^{k}) \geq \sum\nolimits_{h \in v_{ij}^{k} \cap v_{iq}^{k}} t_{iqh}^{k}(Z_{ish}^{k})$$

$$i \in I^{k};\ j, q \in J_{i}^{k} (j < q); k \in K; s \in S_{i}^{k};\ M: large\ number; \quad (5.20)$$

$$C_{is}^{k} \geq X_{isj}^{k}$$

$$i \in I^{k};\ j \in J_{i}^{k};\ s \in S_{i}^{k}; k \in K \quad (5.21)$$

$$X_{isj}^{k}, C_{is}^{k} \geq 0;$$

$$Z_{ish}^{k} \in \{0,1\}; A_{ptis}^{k} \in \{0,1\}\ (p \leq i; p = i \rightarrow t < s); Y_{isjq}^{k} \in \{0,1\}\ (j < q); \emptyset_{isj}^{k} \in \{0,1\}; \quad (5.22)$$

$$i \in G; j \in g_{i}; k \in K; i, p \in I^{k};\ j, q \in J_{i}^{k};\ s \in S_{i}^{k};\ t \in g_{p}^{k}; h \in v_{ij}^{k};\ M: \text{large number}.$$

$$\boldsymbol{SP(m+1), st_{ij(m_{ij}+1)} = m + 1, \forall\ i \in I^{m+1}\ \&\ j \in J_{i}^{m+1}} \quad (5.23)$$

$$Min\ Z_{sp(m)} = \sum_{i\in G}\sum_{j\in g_i}\sum_{s\in g_i}\left(\alpha\times w_{ij} - \left(\sum_{s'\in g_i}\Phi_{ijs\acute{s}(m_{ij})}\right)X_{isj}^{mt}\right)$$
$$+\sum_{i\in G}\sum_{j\in g_i}\left(\beta\times w_{ij}\right)T_{ij}^{t}$$
$$+\sum_{i\in G}\sum_{j\in g_i}\sum_{s\in g_i}\left(\left(\sum_{s'\in g_i}\Phi_{ijs\acute{s}(m_{ij})}\right)M\times\phi_{isj}^{mt}\right)$$
$$+\sum_{i\in G}\sum_{j\in g_i}\sum_{s\in g_i}\left(\left(\sum_{h\in V_{ij}^{k}}\left(\sum_{s'\in g_i}\Phi_{ijs\acute{s}(m_{ij})}\right)t_{ijh}^{m}\right)Z_{ish}^{mt}\right)$$
$$+2\left(\sum_{i\in G}\sum_{j\in g_i}\sum_{s\in g_i}\sum_{s'\in g_i}\Phi_{ijs\acute{s}(m_{ij})}\right)\times M-\psi$$

Subject to:

$$T_{ij} \geq X_{isj}^{m} - d_{ij}$$
$$i \in I^{m+1}, j \in J_i^{m+1};\ s \in S_i^{m}; \tag{5.24}$$

$$X_{isj}^{m}, T_{ij} \geq 0;$$
$$Z_{ish}^{k} \in \{0,1\}; A_{ptis}^{k} \in \{0,1\}\ (p \leq i; p = i \rightarrow t < s); Y_{isjq}^{k} \in \{0,1\}\ (j < q); \emptyset_{isj}^{k} \in \{0,1\}; \tag{5.25}$$
$$i \in G; j \in g_i; k \in K; i,p \in I^{k};\ j,q \in J_i^{k};\ s \in S_i^{k};\ t \in g_p^{k}; h \in v_{ij}^{k};\ M\text{: large number}.$$

A summation of objective function values (5.9), (5.11), and (5.23) is the reduced cost obtained from the DLMP for finding the next promising column to enter the RLMP. Constraint (5.10) ensures that processing of each job cannot be started on the first stage if the job is not released in the initial virtual stage. Set of constraints (5.12) through (5.22) determine independently the partial sequence and assignment of batches on machines as well as the partial sequence of jobs within batches on each machine in each stage. Set of constraints (5.12) through (5.15) determine the optimal batch compositions of groups in a stage. Constraints (5.16) together with constraint (5.17) are incorporated to assign values to binary variables A_{ptis}^{k} and, consequently, find the optimal sequence of batches on machines. Constraint (5.18) ensures that the processing of any job within a batch in each of the stages only starts when the corresponding machine is available to be setup for that batch. Set of constraints (5.19) and (5.20) are incorporated to assign values to binary variables Y_{isjq}^{k} and, consequently, find the optimal sequence of jobs within batches. Constraint (5.21) determines the completion time of each batch. Constraint (5.24) is applied to find the tardiness of each job in the final virtual stage. Finally, constraints (5.22) and (5.25) contains all the signs and binary constraints.

In spite of excluding the combination between batch compositions of all groups for the entire stages as well as bi-criteria objective function and non-zero job release time from the middle stages, each $SP(k)$, $k \in K$ is still a sequence-dependent batching and scheduling problem on either a set of unrelated-parallel machines or one single-machine. Although each SP developed in each node of the B&P algorithm is strongly NP-hard (Du and Leung 1990, Ho and Chang 1995, Karp 1972, Lenstra et al. 1990), the column generation technique together with simplified SPs is developed in each node so that it can be applied even on large-size instances of the batch scheduling problem in HFS. Therefore, DWD2 is introduced in the next section.

5.3.1.2. DWD2

As mentioned before, in the RMILP model, desired lower bound on batch sizes are relaxed from the MILP1 model so that there is a possibility to obtain either an optimal solution or a good quality lower bound for the MILP1 model. Therefore, the batching phase of the MILP1 model is removed and the job assignment and sequence on machines are determined for the entire stages. Since the RMILP model is developed based on the MILP1 model, DWD2 can be considered as a restricted version of DWD1.

Several lifelike assumptions to reflect real industry requirements along with the exhaustive combination enumeration between batch compositions for the entire stages make the problem very complex. As mentioned in DWD1, with the help of two virtual stages, initial and final virtual stages, the assumptions of non-zero job release time as well as bi-criteria objective function are excluded from middle-stage SPs of DWD1. Therefore, the objective function associated with middle-stage SPs is related to only one criterion, i.e., total completion time. All jobs begin their process from the initial virtual stage, while all jobs will be completed at the final virtual stage. Run times of all jobs in these two stages are equal to zero. In addition, an exhaustive combination enumeration between batch compositions for the entire stages is excluded by relaxing the linking constraint. Hence, each $SP(k)$ $\forall k \in K$ is still a batching and scheduling problem on either a single-machine or a set of unrelated-parallel machines.

The identification of structural non-dominance properties corresponding to the batch composition restrictions is a key step to reduce the solution space of middle-stage SPs to a non-dominated set and, consequently, to make possible the solution in affordable time. Therefore, the batching phase of middle-stage SPs of DWD1 is restricted to allocate one and only one job to each batch of each group in DWD2. Thus, the optimal solution is guaranteed when there is no violation on LB_{ih}^{k}; otherwise the SP gives a lower bound for the original SP. This being the case, DWD2 is developed by implementing the following changes on DWD1:

- If there is a desired lower bound(s), relax such requirement(s) as $LB_{ih}^{k} = 1$, where $i \in I^k; h \in v^k; k \in K$;
- Constraints (5.12), (5.14), (5.15), (5.19) and (5.20) are excluded;
- Change variable $\emptyset_{isj}^{k}$ to a parameter as: j^{th} job of group i is assigned to s^{th} batch of that group in each stage, so that $j = s$ (i.e., $\emptyset_{ijj}^{k} = 1$, where $i \in I^k; j \in J_i^k;\ k \in K$; $\emptyset_{ijj}^{k} = 0$, otherwise);
- Change the inequality constraint (5.13) to equality constraint as $\sum_{h \in v^k} Z_{ish}^{k} = 1$; where $i \in I^k; j \in J_i^k;\ s \in S_i^k;\ k \in K$;
- Remove variable Y_{isjq}^{k}, because there is only one job assigned to each batch.

Consequently, the index "s" is removed from decision variables and each middle-stage SP is converted to a job scheduling problem, instead of batch scheduling. Although DWD2 can be developed directly from the RMILP model, it is developed based on DWD1 to show the difference between SPs, particularly middle-stage SPs, of the two developed DWDs.

In the RMILP model, constraint (4.25) is the only linking constraint. With respect to this constraint, completion time of a job in each stage is coupled with the completion time of that job in the preceding stage where the job had its latest operation. By relaxing constraint (4.25), the partial job assignment and sequences on machines related to each stage are determined. In addition to the notations used in Section 4.1.4, the following notations are defined to formulate the RMP and SPs. Subsequently, objective functions and constraint sets of the SPs as well as the RMP and dual of the linear master problem (DLMP) are defined.

$\mathcal{T}$	Set of columns, indexed by t	$\mathcal{T} = \{1,2,\dots,T\}$
T	Total number of columns, where each column represents the partial schedules for all the stages,	
X_{ij}^{kt}	The completion time of job j of group i in stage k in column t	
T_{ij}^{t}	The tardiness of job j of group i in column t	
Z_{ijh}^{kt}	1 if job j of group i is assigned to machine h in stage k in column t; 0 otherwise	
λ_t	1 if column t is selected; 0 otherwise	

Restricted linear master problem (RLMP)

$$Min\ Z = \sum\nolimits_{t \in \mathcal{T}} \left(\sum\nolimits_{i \in G} \sum\nolimits_{j \in g_i} w_{ij} (\alpha . X_{ij}^{\left(st_{ij(m_{ij}+1)}\right)t} + \beta . T_{ij}^{t} \right) \lambda_t \tag{5.26}$$

Subject to:

$$\sum\nolimits_{t\in\mathcal{T}}\left(X_{ij}^{st_{ij(r)}t}-X_{ij}^{st_{ij(r-1)}t}-\sum_{h\in v_{ij}^{st_{ij(r)}}}t_{ijh}^{st_{ij(r)}}\left(Z_{ijh}^{st_{ij(r)}t}\right)\right)\lambda_t\geq 0; \tag{5.27}$$

$$i=1,2,\dots,g; j=1,2,\dots,n_i; r=1,2,\dots,m_{ij},m_{ij}+1;$$

$$\sum\nolimits_{t\in\mathcal{T}}\lambda_t=1; \tag{5.28}$$

$$\lambda_t\geq 0;\ t\in\mathcal{T}. \tag{5.29}$$

Since all variables $X_{ij}^{st_{ij(r)}t}$, $Z_{ijh}^{st_{ij(r)}t}$, and TD_{ij}^{t} related to the RMILP model are assumed to be known, the only decision variable in the model is binary variables λ_t. Assume $\Phi_{ijr}, \forall i=1,2,\dots,g; j=1,2,\dots,n_i; r=1,2,\dots,m_{ij}$ and ψ as dual variables associated with constraints (5.27) and (5.28), respectively.

Subsequently, new objective functions and constraint sets of the SPs as well as the new RMP and DLMP are defined as follows:

Dual of the linear master problem (DLMP)

$$Max\ Z=\psi \tag{5.30}$$

Subject to:

$$\sum\nolimits_{i\in G}\sum\nolimits_{j\in g_i}\sum\nolimits_{r=1}^{m_{ij}+1}\left(X_{ij}^{st_{ij(r)}t}-X_{ij}^{st_{ij(r-1)}t}-\sum_{h\in v_{ij}^{st_{ij(r)}}}t_{ijh}^{st_{ij(r)}}\left(Z_{ijh}^{st_{ij(r)}t}\right)\right)\times\Phi_{ijr}+\psi$$
$$\leq\sum\nolimits_{i\in G}\sum\nolimits_{j\in g_i}w_{ij}\left(\alpha.X_{ij}^{\left(st_{ij\left(m_{ij}+1\right)}\right)t}+\beta.T_{ij}^{t}\right);\ t\in\mathcal{T}; \tag{5.31}$$

$$\Phi_{ijr}\geq 0;\forall i=1,2,\dots,g; j=1,2,\dots,n_i; r=1,2,\dots,m_{ij}; \tag{5.32}$$

$$\psi\quad unrestricted; \tag{5.33}$$

Sub-problems (SPs)

$\boldsymbol{SP(0), st_{ij(0)} = 0, \forall\, i \in I^0 \;\&\; j \in J_i^0}$

$$Min\ Z_{sp(1)} = \sum\nolimits_{i\in G} \sum\nolimits_{j\in g_i} (\Phi_{ij1}) X_{ij}^{0t} \tag{5.34}$$

Subject to:

$$X_{ij}^0 \geq r_{ij};\ i \in I^0, j \in J_i^0. \tag{5.35}$$

$\boldsymbol{SP(k), 1 \leq k \leq m, for\ st_{ij(r)} = k, \forall i = 1,2,\dots,g \;\&\; j = 1,2,\dots,n_i \;\&\; r = 1,2,\dots,m_{ij}}$

$$\begin{aligned} Min\ Z_{sp(k)} = &\sum\nolimits_{i\in G} \sum\nolimits_{j\in g_i} \left(\Phi_{ij(r+1)}\right) X_{ij}^{kt} - \sum\nolimits_{i\in G} \sum\nolimits_{j\in g_i} \left(\Phi_{ijr}\right) X_{ij}^{kt} \\ &+ \sum\nolimits_{i\in G} \sum_{h\in v_{ij}^k} \left(\sum\nolimits_{j\in g_i} t_{ijh}^k \left(\Phi_{ijr}\right)\right) Z_{ijh}^{kt} \end{aligned} \tag{5.36}$$

Subject to:

$$\sum\nolimits_{h\in v_{ij}^k} Z_{ijh}^k = 1; i \in I^k;\ j \in J_i^k; \tag{5.37}$$

$$\begin{gathered} X_{ij}^k + M\left(1 - A_{pj'ij}^k\right) + M\left(1 - Z_{ijh}^k\right) + M\left(1 - Z_{pj'h}^k\right) \geq X_{pj'}^k + S_{pih}^k + t_{ijh}^k \\ i,p \in I^k\ (p \leq i; p = i \rightarrow j \neq j');\ j' \in J_p^k;\ j \in J_i^k; h \in v_{ij}^k \cap v_{pj'}^k; \end{gathered} \tag{5.38}$$

$$\begin{gathered} X_{pj'}^k + M\left(A_{pj'ij}^k\right) + M\left(1 - Z_{ijh}^k\right) + M\left(1 - Z_{pj'h}^k\right) \geq X_{ij}^k + S_{iph}^k + t_{pj'h}^k \\ i,p \in I^k\ (p \leq i; p = i \rightarrow j \neq j');\ j' \in J_p^k;\ j \in J_i^k; h \in v_{ij}^k \cap v_{pj'}^k; \end{gathered} \tag{5.39}$$

$$X_{ij}^k \geq \sum\nolimits_{h\in v_{ij}^k} \left(a_h^k + S_{0ih}^k + t_{ijh}^k\right) Z_{ijh}^k\ ;\ i \in I^k;\ j \in J_i^k; \tag{5.40}$$

$$\begin{gathered} X_{ij}^k, TD_{ij} \geq 0;\ Z_{ijh}^k \in \{0,1\}; A_{pj'ij}^k \in \{0,1\}\ (p \leq i; p = i \rightarrow j \neq j'); i,p \in I^k; j \in J_i^k; j' \in \\ J_p^k; h \in v_{ij}^k. \end{gathered} \tag{5.41}$$

$\boldsymbol{SP(m+1), st_{ij(m_{ij}+1)} = m+1, \forall\, i \in I^{m+1}}$ & $\boldsymbol{j \in J_i^{m+1}}$

$$Min\ Z_{sp(m)} = \sum_{i\in G}\sum_{j\in g_i}(\alpha * w_{ij} - \Phi_{ijm})X_{ij}^{mt} + \sum_{i\in G}\sum_{j\in g_i}(\beta * w_{ij})T_{ij}^{t} + \sum_{i\in G}\sum_{h\in v_{ij}^{m}}\left(\sum_{j\in g_i} t_{ijh}^{m}(\Phi_{ijr})\right)Z_{ijh}^{mt} - \psi \tag{5.42}$$

Subject to:

$$T_{ij} \geq X_{ij}^{m} - d_{ij};\ i \in I^{m+1}, j \in J_i^{m+1}; \tag{5.43}$$

$$X_{ij}^{m}, T_{ij} \geq 0;\ Z_{ijh}^{m} \in \{0,1\}; A_{pj\prime ij}^{m} \in \{0,1\}\ (p \leq i);\ i,p \in I^{m+1}; j \in J_i^{m+1}; h \in v_{ij}^{m+1}. \tag{5.44}$$

Set of constraints (5.26) through (5.29) and set of constraints (5.30) through (5.33) determine the RLMP and DLMP, respectively, with respect to simplifications to the SPs. A summation of objective function values (5.34), (5.36), and (5.42) is the reduced cost obtained from the DLMP for finding the next promising column to enter the RLMP. Constraint (5.35) ensures that processing of each job cannot be started on the first stage if the job is not released in the initial virtual stage. Set of constraints (5.37) through (5.41) determine independently the partial sequence of jobs belonging to different groups on each machine in each stage. Constraint (5.37) ensures that each job of a group is assigned to only one machine. Constraints (5.38) together with constraint (5.39) are incorporated to find the sequence of jobs. With respect to these constraints, if two jobs are processed on the same machine, the completion time of the succeeding job (which is not skipping the stage) must be greater than the completion time of the preceding job, plus the sequence-dependent setup time and the run time required for processing the job on a particular stage. *Simultaneously*, these constraints assign values to binary variables $A_{pj\prime ij}^{k}$, which determine the sequence of jobs on the same machine in each stage. Constraint (5.40) ensures that the processing of any job in each of the stages only starts when the corresponding machine is available to be setup for that job. Constraint (5.43) is applied to find the tardiness of each job in the final virtual stage. Finally, constraints (5.41) and (5.44) contains all the signs and binary constraints.

Since DWD2 is based on the RMILP model, which itself is the restricted version of the MILP1 model, the lower bound obtained by DWD2 might be a tight lower bound for the RMILP model unless the number of violations in LB_{ih}^{k} is not significant (case 6 in Figure 8). In this case, given the computational time limit, there is a possibility of obtaining a tight lower bound for any of the MILP models by DWD2, particularly for medium- and large-size problems.

5.3.1.3. DWD3

Since middle-stage SPs of DWD1 are NP-hard, this algorithm might not be able to present tight lower bounds, particularly for large-size problems. In addition, DWD2 can obtain tight lower bounds for any of the MILP models only when the middle-stage SPs of this algorithm do not violate LB_{ih}^{k} significantly. Therefore, similar to the structure of DWD1, DWD3 is developed based on the MILP3 model so that it guarantees tight lower bounds of problems, as LB_{ih}^{k} will never be violated.

In the MILP3 model, constraint (4.43) is the only linking constraint. By relaxing this constraint, all the other constraints include only the decision variables corresponding to a particular stage at a time and, consequently, there is one SP for each stage, determining the partial job assignment and sequence on that stage. These partial sequences are linked together with the help of constraint (4.43) in the RMP. Apart from this, two virtual stages are implemented to simplify middle-stage SPs. In addition to the notations used in Section 4.1.3 and following the same procedure presented in Section 5.3.1.1, the following notations are defined to formulate the RMP and SPs. Subsequently, objective functions and constraint sets of the SPs as well as the RMP and dual of the linear master problem (DLMP) are defined.

$\mathcal{T}$	Set of columns, indexed by t	$\mathcal{T} = \{1,2,\ldots,T\}$
T	Total number of columns, where each column represents the partial schedules for all the stages	
X_j^{kt}	The completion time of job j in stage k in column t	
T_j^t	The tardiness of job j in column t	
x_{ljh}^{kt}	1 job j is scheduled immediately after job l on machine h in stage k in column t; 0 otherwise	
λ_t	1 if column t is selected; 0 otherwise	

Restricted linear master problem (RLMP)

$$Min\ Z = \sum_{t\in\mathcal{T}}\left(\sum_{j\in N} w_j(\alpha.X_j^{\left(st_{j(m_j+1)}\right)t} + \beta.T_j^t\right)\lambda_t \tag{5.45}$$

Subject to:

$$\sum_{t\in\mathcal{T}}\left(X_j^{st_{j(r)}t} - X_j^{st_{j(r-1)}t} - \sum_{\substack{l\in N^{st_{j(r)}}\cup\{0\}\\ j\neq l}} \sum_{h\in V_l^{st_{j(r)}}\cap V_j^{st_{j(r)}}} t_{jh}^{st_{j(r)}}\left(x_{ljh}^{st_{j(r)}t}\right)\right)\lambda_t \geq 0; \tag{5.46}$$

$$\forall j \in N; r = 1,2,\dots,m_{ij},m_{ij}+1;$$

$$\sum_{t\in\mathcal{T}}\lambda_t = 1; \tag{5.47}$$

$$\lambda_t \geq 0;\ t\in\mathcal{T}. \tag{5.48}$$

Assume $\Phi_{jr}, \forall j \in N; r = 1,2,\dots,m_{ij}$ and ψ as dual variables associated with constraint (5.46) and (5.47), respectively. Subsequently, new objective functions and constraint sets of the SPs as well as the new RMP and DLMP are defined as follows:

Dual of the linear master problem (DLMP)

$$Max\ Z = \psi \tag{5.49}$$

Subject to:

$$\sum_{j\in N}\sum_{r=1}^{m_{ij}+1}\left(X_j^{st_{j(r)}t} - X_j^{st_{j(r-1)}t} - \sum_{\substack{l\in N^{st_{j(r)}}\cup\{0\}\\ j\neq l}} \sum_{h\in V_l^{st_{j(r)}}\cap V_j^{st_{j(r)}}} t_{jh}^{st_{j(r)}}\left(x_{ljh}^{st_{j(r)}t}\right)\right)\times\Phi_{jr} + \psi \tag{5.50}$$

$$\leq \sum_{j\in N} w_j(\alpha.X_j^{\left(st_{j(m_j+1)}\right)t} + \beta.T_j^t\ ;\ t\in\mathcal{T};$$

$$\Phi_{jr} \geq 0; \forall j \in N; r = 1,2,\dots,m_{ij}; \tag{5.51}$$

$$\psi \quad unrestricted; \tag{5.52}$$

Sub-problems (SPs)

$\boldsymbol{SP(0), st_{j(0)} = 0,\ \& \ j \in N}$

$$Min\ Z_{sp(1)} = \sum\nolimits_{j \in N} (\Phi_{j1}) X_j^{0t} \tag{5.53}$$

Subject to:

$$X_j^0 \geq r_j;\ j \in N. \tag{5.54}$$

$\boldsymbol{SP(k), 1 \leq k \leq m, for\ st_{j(r)} = k, \forall j \in N \ \& \ r = 1, 2, \ldots, m_{ij}}$

$$\begin{aligned} Min\ Z_{sp(k)} &= \sum\nolimits_{j \in N} \left(\Phi_{j(r+1)}\right) X_j^{kt} - \sum\nolimits_{j \in N} \left(\Phi_{jr}\right) X_j^{kt} \\ &+ \sum\nolimits_{\substack{l \in N^k \cup \{0\} \\ j \neq l}} \sum_{h \in V_l^k \cap V_j^k} \left(\sum\nolimits_{j \in N} t_{jh}^k \left(\Phi_{jr}\right) \right) x_{ljh}^{kt} \end{aligned} \tag{5.55}$$

Subject to:

$$\sum_{\substack{l \in N^k \cup \{0\} \\ l \neq j}} \sum_{h \in V_l^k \cap V_j^k} x_{ljh}^k = 1 \qquad \forall j \in N^k; \tag{5.56}$$

$$\sum_{j \in N^k} x_{0jh}^k \leq 1 \qquad \forall h \in V^k; \tag{5.57}$$

$$\sum_{\substack{j \in N^k \\ l \neq j}} \sum_{h \in V_l^k \cap V_j^k} x_{ljh}^k \leq 1 \qquad \forall l \in N^k; \tag{5.58}$$

$$\sum_{\substack{l \in N \cup \{0\} \\ l \neq j}} x_{ljh}^{k} = \sum_{\substack{l \in N \cup \{n+1\} \\ l \neq j}} x_{jlh}^{k} \qquad (5.59)$$

$$\forall j \in N^{k}, \forall h \in V_{j}^{k};$$

$$\left(\sum_{j=j_1}^{j_\ell} \sum_{l \in g_i | l \neq j_1, l \neq j_2, \dots, l \neq j_\ell} \left(x_{ljh}^{k} + x_{jlh}^{k}\right)\right) + 2\left(\sum_{j=j_1}^{j_\ell} \sum_{j'=j+1}^{j_\ell} \left(x_{jj'h}^{k} + x_{j'jh}^{k}\right)\right) \geq 2\left(Y_{j_1 j_2 \dots j_l}^{ihk}\right)\left(LB_{ih}^{k} - 1\right) \qquad (5.60)$$

$$M\left(Y_{j_1 j_2 \dots j_l}^{ihk}\right) \geq \sum_{j=j_1}^{j_\ell} \sum_{j'=j+1}^{j_\ell} \left(x_{jj'h}^{k} + x_{j'jh}^{k}\right) \qquad (5.61)$$

$$M\left(Y_{j_1 j_2 \dots j_l}^{ihk} - 1\right) \leq \sum_{j=j_1}^{j_\ell} \sum_{j'=j+1}^{j_\ell} \left(x_{jj'h}^{k} + x_{j'jh}^{k}\right) - \varepsilon\left(Y_{j_1 j_2 \dots j_l}^{ihk}\right) \qquad (5.62)$$

$$\forall h \in V^{k}, \forall i \in I^{k} | LB_{ih}^{k} = \ell \ (\ell > 1), \forall \{j_1, j_2, \dots, j_l\} \in Q_{ih}^{k}, 0 < \varepsilon < 1;$$

$$X_{j}^{k} + M\left(1 - x_{ljh}^{k}\right) \geq X_{l}^{k} + S_{ljh}^{k} + t_{jh}^{k} \qquad (5.63)$$

$$\forall l \in N^{k} \cup \{0\}, \forall j \in N^{k} | l \neq j, \forall h \in V_{l}^{k} \cap V_{j}^{k};$$

$$X_{0}^{k} = a_{h}^{k}$$

$$\forall h \in V^{k};$$

$$X_{j}^{k} \geq 0 \qquad (5.64)$$

$$\forall j \in N^{k};$$

$$x_{ljh}^{k} \in \{0, 1\}$$

$$\forall l \in N^{k} \cup \{0\}, \forall j \in N^{k} \cup \{n+1\} | l \neq j, \forall h \in V_{l}^{k} \cap V_{j}^{k}.$$

$\boldsymbol{SP(m+1), st_{j(m_j+1)} = m+1, \forall\, j \in N}$

$$Min\ Z_{sp(m)} = \sum\nolimits_{j\in N}(\alpha * w_j - \Phi_{jm})X_j^{mt} + \sum\nolimits_{j\in N}(\beta * w_j)T_j^t + \sum\nolimits_{\substack{l\in N^m\cup\{0\}\\ j\neq l}} \sum_{h\in V_l^m \cap V_j^m} \left(\sum\nolimits_{j\in N} t_{jh}^m(\Phi_{jr})\right) x_{ljh}^{mt} - \psi \tag{5.65}$$

Subject to:

$$T_j \geq X_j^m - d_j;\ \forall j \in N; \tag{5.66}$$

$$X_j^m, T_j \geq 0;\ x_{ljh}^m \in \{0,1\}; l, j \in N^{m+1};\quad h \in V_l^{m+1} \cap V_j^{m+1}. \tag{5.67}$$

Set of constraints (5.45) through (5.48) and set of constraints (5.49) through (5.52) determine the RLMP and DLMP, respectively, with respect to simplifications for the SPs. A summation of objective function values (5.49), (5.53), and (5.65) is the reduced cost obtained from the DLMP for finding the next promising column to enter the RLMP. Constraint (5.54) ensures that processing of each job cannot be started on the first stage if the job is not released in the initial virtual stage. Set of constraints (5.56) through (5.64) determine independently the partial sequence of jobs belonging to different groups on each machine in each stage, regarding the desired lower bounds on batch sizes. Set of constraints (5.56) through (5.59) determine the sequence of jobs on machines. Set of constraints (5.60) through (5.62) consider the desired lower bounds on batch sizes. As a result, set of constraints (5.56) through (5.62) determine the optimal job sequence on machines regarding the desired lower bounds on batch sizes. Constraints (5.63) is incorporated to find the completion time of jobs. Constraint (5.66) is applied to find the tardiness of each job in the final virtual stage. Finally, constraints (5.64) and (5.67) contains all the signs and binary constraints.

5.3.1.4. Sub-problem acceleration time

The SPs developed by DWDs should be optimized at each iteration of column generation in order to identify columns with the most negative reduced costs and introduce the best one to the master problem. However, not only the column with the most negative reduced cost, but also *any* column with a negative reduced cost is a candidate to improve the RMP during the course of column generation. Some researchers (Barnhart et al. 1998, Vanderbeck 1994) have suggested that a heuristic algorithm can be used in early stages of column generation algorithm to approximately solve a SP for identifying new columns with negative reduced costs, especially if they are computationally too expensive to solve. Then, whenever the heuristic fails to find a promising column, the search should be switched to an exact algorithm to optimally solve the SP. Therefore, a heuristic algorithm can be used to approximately solve the SPs to identify new columns with negative reduced costs, especially if they are computationally too expensive to solve. With respect to solving the SPs heuristically, two approaches corresponding to introducing columns to the master problem exist as follows:

1. Stop a heuristic algorithm as soon as a negative reduced cost column is identified.
2. Select all negative reduced cost columns that the heuristic identifies and append them all into the RMP.

The first approach will reduce the computation time per iteration but the overall effect may not be attractive since the number of iterations will probably increase, while the second approach will require more time per iteration and may decrease the total number of iterations. Since more columns are added to the RMP in each iteration, the RMP grows rapidly and may become harder to solve.

The lower bounds are valid only when all the SPs are solved optimally and exact optimization is applied. An optimal solution of an SP cannot be guaranteed by heuristic procedure. Apart from this, there may exist negative reduced cost columns but the heuristic is not able to identify them. In order to deal with this problem, a two-phase approach is employed to solve the SPs. In the first phase, a fast heuristic is used to solve the SP approximately as long as it identifies a negative reduced cost column. In case the heuristic fails to identify such a column, an exact algorithm that solves the SP to optimality is invoked to prove optimality or to generate a column with a negative reduced cost in the second stage. Although the column generation algorithm is initialized by either a solution obtained heuristically or the developed initial solution finding mechanism, this acceleration technique to solve the SPs is more beneficial in solving the SPs obtained from DWD1 as the SPs obtained from DWD2 and DWD3 might be optimally solved, particularly for small- and medium-size problems.

5.3.1.5. Comparison between developed DWDs

During the iterative process of column generation for a small-, medium-, and large-size problem, the progress of DWDs are demonstrated in Figures 21 through 23, respectively, and the following results are obtained from comparison between DWDs:

- The lower bound obtained by DWD3 present the minimum gap from either the optimal solution or the upper bound compared to the lower bounds obtained by DWD1 and DWD2 for all problem sizes.

- Since the number of violations on batch sizes is not significant, DWD2 is capable of identifying a lower bound with 4.5% deviation from the lower bound developed by DWD3 for a small-size problem. In addition, DWD2 develops lower bounds with a gap of around 21.2% and 13.5% from the lower bounds identified by DWD3 for medium- and large-size problems because the number of violations on batch sizes is significant.
- The lower bounds obtained by CPLEX are very poor in quality compared to the upper bounds obtained by PSO/LSA. Alternatively, DWD3 establishes good quality lower bounds as a baseline for evaluation of the proposed algorithms.
- The lower bound developed by DWD1 has a large gap compared to the lower bounds developed by DWD3, particularly for the large-size problem, because the SPs obtained by DWD1 are still so large that they cannot be optimally solved.
- Although DWD2 converges to the upper bound faster than DWD3 in early iterations of column generation, there is a significant gap between the lower bounds developed by DWD2 and DWD3 because the SPs obtained by DWD2 violate significantly the desired lower bounds on batch sizes.

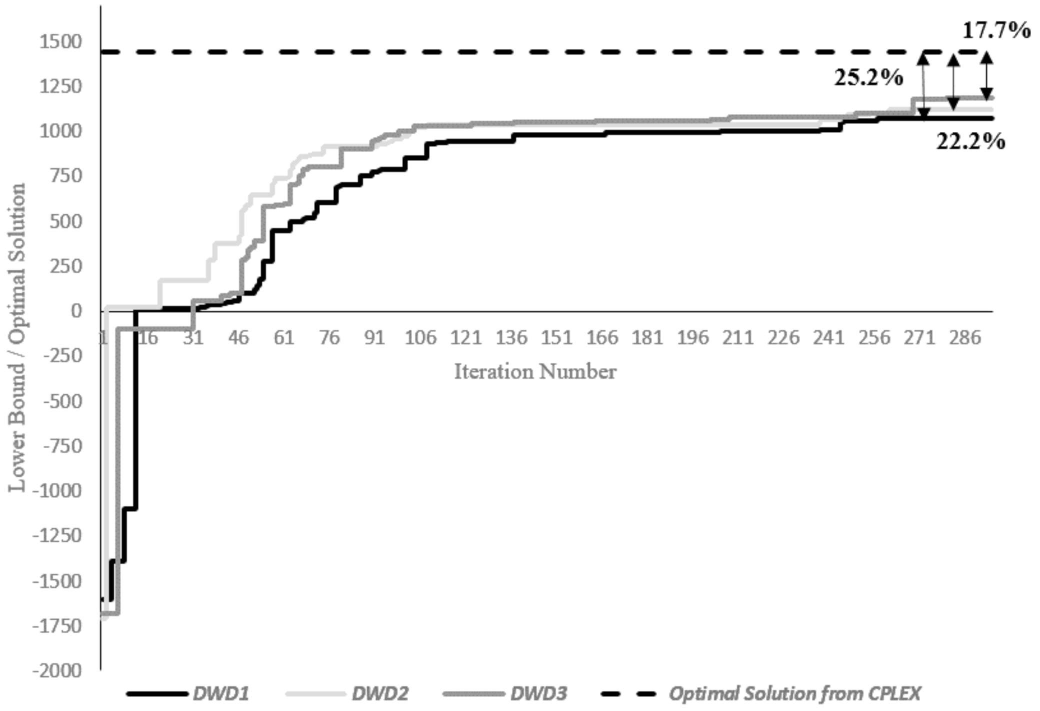


Figure 21. Iterative progress of DWDs on a small-size problem

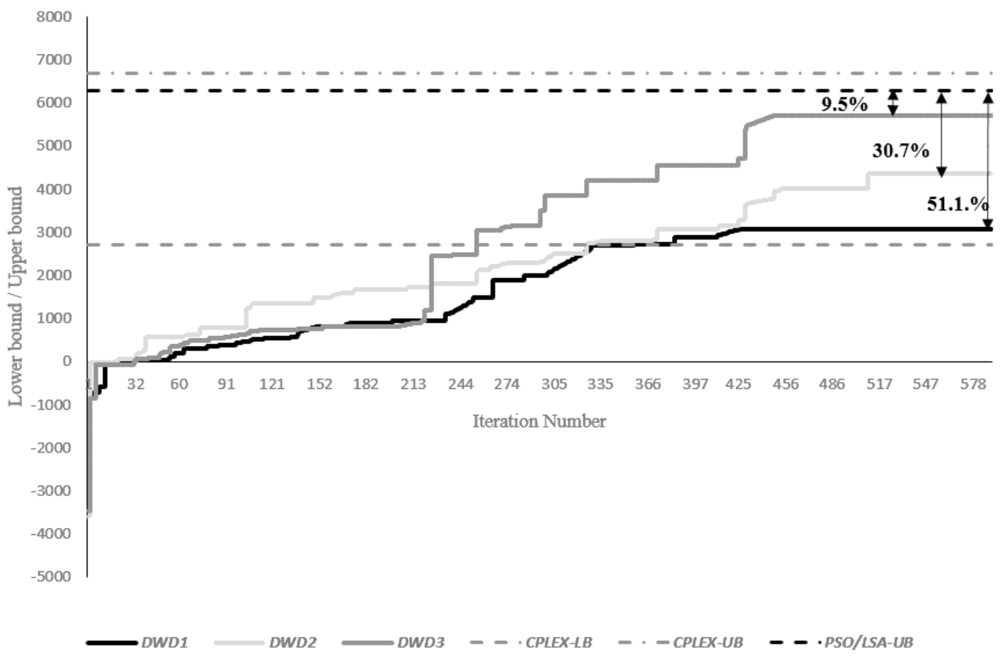


Figure 22. Iterative progress of DWDs on a medium-size problem

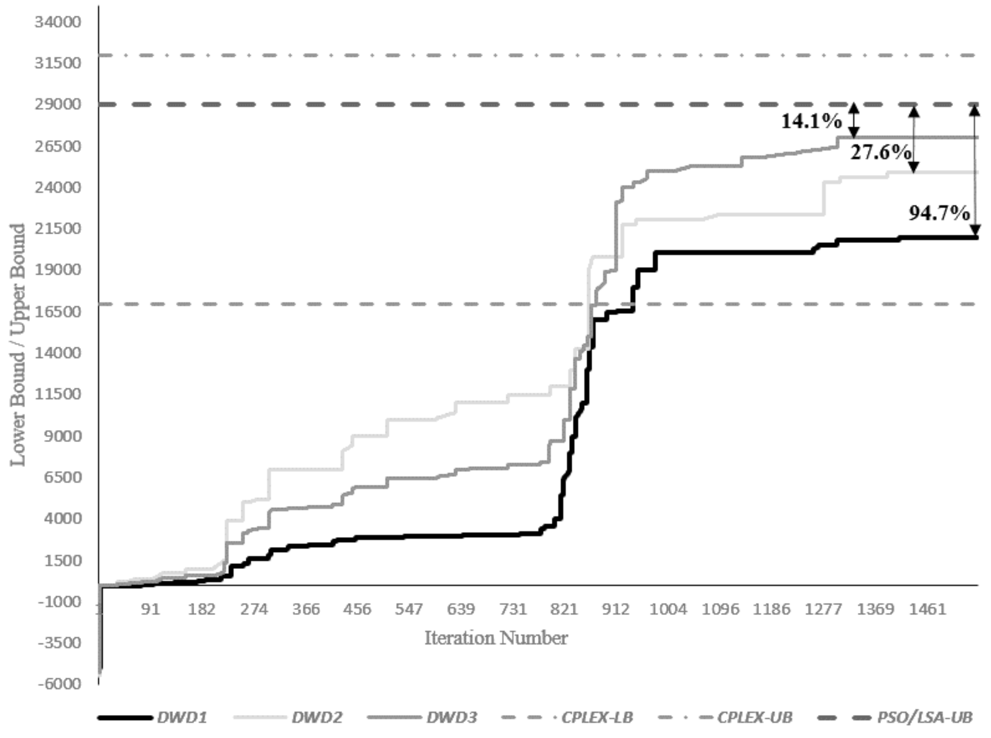


Figure 23. Iterative progress of DWDs on a large-size problem

In conclusion, our preliminary experiments revealed that the SPs obtained by the DWD3 decomposition technique can be optimally solved, while DWD1 is only capable of finding lower bounds for small-size problems. Also, for larger size of problems, the SPs obtained by DWD1 are still NP-hard so that they cannot be optimally solved, even when the acceleration technique is applied for developed SPs. Therefore, DWD1 is not capable of finding even good quality lower bound for large-size problems. In addition, the SPs obtained by DWD2 can be optimally solved but the desired lower bounds on batch sizes might be violated by some of the SPs. Therefore, DWD2 cannot guarantee finding tight lower bounds if the number of violations in LB_{ih}^{k} is significant. Thus, DWD2 and DWD3 are the only approaches that have been used in the B&P algorithm.

5.3.2. Stabilization

Barnhart et al. (1998) noted that column generation has been successfully applied to many large size problems, particularly in the field of routing and scheduling. However, this approach is known for its slow convergence. In other words, remarkable improvement is achieved in a very short period of time, only during the first iterations of column generation, while little progress is obtainable when the search is close to the optimal solution in later iterations. This phenomenon is referred to as the *tailing-off* effect, and usually causes adverse impacts on the efficiency of the algorithm (Lübbecke and Desrosiers 2005).

It is possible that the reduced costs of promising columns are not appropriately estimated by dual values in early iterations of column generation. The dual values obtained by the DLMP are the extreme points of the dual polyhedron, which are characterized by very large values of some columns while the others are set to zero. Therefore, the reduced costs estimated by those extreme points oscillate severely, which lead the algorithm to perform many more iterations to find the optimal solution of the RLMP.

In order to accelerate the convergence of column generation, several techniques, which are generally referred to as *stabilization*, are proposed in the literature in order to accelerate the convergence of column generation. The main idea behind all techniques lies on the fact that these techniques prevent the dual variables from taking extreme values. Marsten et al. (1975) proposed one of the earliest stabilization techniques, referred to as BOXSTEP. Kim et al. (1995) pursue the same purpose, but used a different approach. Du Merle et al. (1997) proposed a very well-known stabilization technique, which combines the concepts of both the approaches proposed by Marsten et al. (1975) and Kim et al. (1995), referred to BOXPEN. Also, Rousseau et al. (2007) proposed a totally different stabilization technique compared to the approach proposed by Du Merle et al. (1997). In this research, the BOXPEN technique is implemented to stabilize column generation since it is shown to be very effective in reducing the total computational time (Bozorgirad 2013). Based on this technique, a soft box is considered for each dual variable to prevent them from taking extreme values. In the following sub-sections, the BOXPEN stabilization method is implemented for DWD2 and DWD3 so that some modifications must be made in the RLMP and DLMP while there is no modification needed for the SPs.

5.3.2.1. Stabilizing DWD2

$\boldsymbol{LMP^{Restricted}}$

$$Min\ Z = \sum_{t\in\mathcal{T}} \left(\sum_{i\in G} \sum_{j\in g_i} w_{ij} (\alpha . X_{ij}^{\left(st_{ij(m_{ij}+1)}\right)t} + \beta . T_{ij}^{t} \right) \lambda_t + \sum_{i\in G} \sum_{j\in g_i} \sum_{r=1}^{m_{ij}+1} \left(-\delta_{ijr}^{-} . \xi_{ijr}^{-} + \delta_{ijr}^{+} . \xi_{ijr}^{+}\right) \tag{5.68}$$

Subject to:

$$\sum_{t\in\mathcal{T}} \left(X_{ij}^{st_{ij(r)}t} - X_{ij}^{st_{ij(r-1)}t} - \sum_{h\in v_{ij}^{st_{ij(r)}}} t_{ijh}^{st_{ij(r)}} \left(Z_{ijh}^{st_{ij(r)}t} \right) \right) \lambda_t - \xi_{ijr}^{-} + \xi_{ijr}^{+} \geq 0; \tag{5.69}$$
$$i = 1,2,\dots,g; j = 1,2,\dots,n_i; r = 1,2,\dots,m_{ij}, m_{ij}+1;$$

$$\sum_{t\in\mathcal{T}} \lambda_t = 1; \tag{5.70}$$

$$-\xi_{ijr}^{-} \geq -\varepsilon_{ijr}^{-} \tag{5.71}$$

$$-\xi_{ijr}^{+} \geq -\varepsilon_{ijr}^{+} \tag{5.72}$$

$$\lambda_t \geq 0; \xi_{ijr}^{-} \geq 0;\ \xi_{ijr}^{+} \geq 0; \tag{5.73}$$
$$t \in \mathcal{T}; i = 1,2,\dots,g; j = 1,2,\dots,n_i; r = 1,2,\dots,m_{ij}, m_{ij}+1.$$

where ξ_{ijr}^{-} and ξ_{ijr}^{+} are the BOXPEN artificial variables so that the term $\sum_{i\in G}\sum_{j\in g_i}\sum_{r=1}^{m_{ij}+1}\left(-\delta_{ijr}^{-} . \xi_{ijr}^{-} + \delta_{ijr}^{+} . \xi_{ijr}^{+}\right)$ in the objective function of the LMP penalizes theses variables. In order to create limits on the dual variables, constraint (5.69) is modified to include ξ_{ijr}^{-} and ξ_{ijr}^{+}. In addition, constraints (5.71) and

(5.72) define ε_{ijr}^{-} and ε_{ijr}^{+} as upper bounds on ξ_{ijr}^{-} and ξ_{ijr}^{+}, respectively. The following new dual variables are defined:

ϱ_{ijr}^{-}	Dual variable associated with constraint (5.71)
ϱ_{ijr}^{+}	Dual variable associated with constraint (5.72)

DLMP^Stabilized

$$Max\ Z = \psi + \sum_{i\in G}\sum_{j\in g_i}\sum_{r=1}^{m_{ij}+1}\left(-\varepsilon_{ijr}^{-}.\varrho_{ijr}^{-} - \varepsilon_{ijr}^{+}.\varrho_{ijr}^{+}\right) \tag{5.74}$$

Subject to:

$$\sum_{i\in G}\sum_{j\in g_i}\sum_{r=1}^{m_{ij}+1}\left(X_{ij}^{st_{ij(r)}t} - X_{ij}^{st_{ij(r-1)}t} - \sum_{h\in v_{ij}^{st_{ij(r)}}} t_{ijh}^{st_{ij(r)}}\left(Z_{ijh}^{st_{ij(r)}t}\right)\right)\times\Phi_{ijr} + \psi$$
$$\le \sum_{i\in G}\sum_{j\in g_i} w_{ij}\left(\alpha.X_{ij}^{\left(st_{ij(m_{ij}+1)}\right)t} + \beta.T_{ij}^{t}\right);\ t\in\mathcal{T}; \tag{5.75}$$

$$-\Phi_{ijr} - \varrho_{ijr}^{-} \le -\delta_{ijr}^{-} \tag{5.76}$$

$$\Phi_{ijr} - \varrho_{ijr}^{+} \le \delta_{ijr}^{+} \tag{5.77}$$

$$\Phi_{ijr} \ge 0;\ \varrho_{ijr}^{-} \ge 0;\ \varrho_{ijr}^{+} \ge 0; \forall i = 1,2,\dots,g; j = 1,2,\dots,n_i; r = 1,2,\dots,m_{ij}; \tag{5.78}$$

$$\psi \quad unrestricted. \tag{5.79}$$

Constraints (5.76) and (5.77) create limits on the dual variables. However, these variables are allowed to exceed those limits with the assignment of a penalty in the objective function (5.74).

5.3.2.2. Stabilizing DWD3

LMP^Restricted

$$Min\ Z = \sum\nolimits_{t\in\mathcal{T}} \left(\sum\nolimits_{j\in N} w_j(\alpha . X_j^{\left(st_{j(m_j+1)}\right)t} + \beta . T_j^t\right)\lambda_t + \sum\nolimits_{j\in N} \sum_{r=1}^{m_{ij}+1} \left(-\delta_{jr}^- . \xi_{jr}^- + \delta_{jr}^+ . \xi_{jr}^+\right) \tag{5.80}$$

Subject to:

$$\sum\nolimits_{t\in\mathcal{T}} \left(X_j^{st_{j(r)}t} - X_j^{st_{j(r-1)}t} - \sum_{\substack{l\in N^{st_{j(r)}}\cup\{0\} \\ j\neq l}} \sum_{h\in V_l^{st_{j(r)}}\cap V_j^{st_{j(r)}}} t_{jh}^{st_{j(r)}} \left(x_{ljh}^{st_{j(r)}t}\right) \right)\lambda_t - \xi_{jr}^- + \xi_{jr}^+ \geq 0; \tag{5.81}$$

$$\forall j \in N; r = 1,2,\ldots,m_{ij}, m_{ij}+1;$$

$$\sum\nolimits_{t\in\mathcal{T}} \lambda_t = 1; \tag{5.82}$$

$$-\xi_{jr}^- \geq -\varepsilon_{jr}^- \tag{5.83}$$

$$-\xi_{jr}^+ \geq -\varepsilon_{jr}^+ \tag{5.84}$$

$$\lambda_t \geq 0; \xi_{jr}^- \geq 0;\ \xi_{jr}^+ \geq 0;$$

$$\forall t \in \mathcal{T};\ j \in N; r = 1,2,\ldots,m_{ij}, m_{ij}+1. \tag{5.85}$$

Define the following new dual variables:

ϱ_{jr}^-	Dual variable associated with constraint (5.83)
ϱ_{jr}^+	Dual variable associated with constraint (5.84)

DLMP^{Stabilized}

$$Max\ Z = \psi + \sum\nolimits_{j\in N} \sum_{r=1}^{m_{ij}+1} \left(-\varepsilon_{jr}^- . \varrho_{jr}^- - \varepsilon_{jr}^+ . \varrho_{jr}^+\right) \tag{5.86}$$

Subject to:

$$\sum_{j\in N}\sum_{r=1}^{m_{ij}+1}\left(X_j^{st_{j(r)}t}-X_j^{st_{j(r-1)}t}-\sum_{\substack{l\in N^{st_{j(r)}}\cup\{0\}\\ j\neq l}}\ \sum_{h\in V_l^{st_{j(r)}}\cap V_j^{st_{j(r)}}} t_{jh}^{st_{j(r)}}\left(x_{ljh}^{st_{j(r)}t}\right)\right)\times\Phi_{jr}+\psi \tag{5.87}$$

$$\leq\sum_{j\in N} w_j(\alpha.X_j^{\left(st_{j(m_j+1)}\right)t}+\beta.T_j^t\ ;\ t\in\mathcal{T};$$

$$-\Phi_{jr}-\varrho_{jr}^{-}\leq-\delta_{jr}^{-} \tag{5.88}$$

$$\Phi_{jr}-\varrho_{jr}^{+}\leq\delta_{jr}^{+} \tag{5.89}$$

$$\Phi_{jr}\geq 0;\varrho_{jr}^{-}\geq 0;\ \varrho_{jr}^{+}\geq 0;\ \forall j\in N;r=1,2,\dots,m_{ij}; \tag{5.90}$$

$$\psi\quad unrestricted. \tag{5.91}$$

5.3.3. Early termination of column generation

In spite of implementing stabilization technique to reduce the number of iterations required by column generation, the difficulties of tailing-off effects do not completely resolve. In other words, there will still be large number of columns required to optimally solve the linear master problem (LMP). Therefore, instead of solving the LMP to optimality, we can decide to permanently end the column generation progress and work with the bounds on the final LMP (Barnhart et al. 1998). Farley (1990), Lasdon (1970), Vanderbeck and Wolsey (1996) describe different methods for calculating such a bound in column generation. The bound is determined as follows:

Z^*	The optimal solution of the LMP
$\bar{Z}$	The optimal solution of the RLMP
${Z^*}_{sp^k}$	The most negative reduced cost corresponding to the SP k

Lübbecke and Desrosiers (2005) describe that when there is an upper bound on the variables of LMP such as $\sum_{j\epsilon J}\lambda_j\leq\zeta$, where J is the set of all possible columns, not only an upper bound, but also a lower bound can be established for the optimal solution of the LMP. This bound is presented as $\bar{Z}+\sum_{k\epsilon K}{Z^*}_{sp}\leq Z^*\leq\bar{Z}$, where k is the set of all SPs.

5.3.4. Branching rule

Since DWD2 and DWD3 are used to decompose the original problem into SPs and DWD3 presents a better performance compared to DWD2, the following branching rule is explained in detail for the B&P algorithm developed based on DWD3. The following three important decisions should be made in relation to branching:

- variable selection amongst different kinds of variables for branching
- choice order for branching variables
- variable selection amongst the same kind of variables for branching
- node selection for branching

Combining column generation with a B&B algorithm, known as B&P algorithm, is not straightforward since branching on only the integer variables of the MP, i.e., λ, leads to an inefficient B&P algorithm. The reason lies on the fact that branching on the λ variables does not provide an efficient strategy for restricting the number of newly generated columns since there is an unlimited number of columns that can be introduced to the MP by branching on each node of the B&B tree. A remedy to this difficulty is to perform branching on integer variables of the MILP3. An integrality constraint on one of the original variables, such as x, requires that $\sum_{t\in T} x_t\lambda_t = 1$ be integral, where T is the set of all columns in the RMP. This constraint is equivalent to $\lambda \in \{0, 1\}$ in the RMP model (Bozorgirad 2013). Thus, instead of branching directly on the λ variables, the integer variables of the original model are participating in the branching scheme. The only two integer variables of the MILP3 are binary variables and they are divided into the following two sets:

- x-variables that determine the assignment and sequence of jobs on machines
- Y-variables that determine a sequence of jobs of the same group processed consecutively on machines

The choice order for branching has an impact on the efficiency of the B&P algorithm so that the important decisions (which lead to other decisions be made automatically) must be made in early stages of the B&P tree. If this were the case, the assignment and sequence of jobs on machines can have the most significant impact on completion time of jobs. Therefore, the x-variables are selected as the first choice for branching. Since a tight lower bound is desirable, particularly for large-size problems, the process of branching can be terminated anywhere in the tree and the lower bound is determined with the help of the B&B algorithm.

With respect to the priority of the set of variables selected for branching, i.e., x-variables or Y-variables, an integer variable should be selected for further branching. In order to select a variable amongst the same kind, i.e., x-variables or Y-variables, on which to perform branching, the value of $\sum_{t\in T} x_t\lambda_t = 1$ is calculated for all integer variables so that the one which is closest to 0.5 is selected as the branching variable. In relation with the node selection for further branching, a depth-first strategy is used for selecting the node with the minimum objective function amongst all possible nodes for branching. In this case, any improvement in the objective function of this node will directly improve the lower bound obtained by the B&B tree.

In conclusion, the B&P algorithm is known to be very time consuming. Therefore, it is of vital importance to gain the most improvement possible by branching on the minimum number of nodes. The following description explains how a high contribution of x-binary variables in the objective function of the MP can reduce the total number of branches. The binary variables x_{ljh}^{k} are interdependent on the constraints (4.33) through (4.36) so that knowing the value of one of them will enable determining the value of the rest. For example, if job 2 is assigned to machine 3 in stage 4 when job 1 is the predecessor of job 2, and there are three machines in the fourth stage of HFS, then it is known that $x_{123}^{4} = 1$, while $x_{l2h}^{4} = 0$, $\forall l \in N^4 - \{1, 2\}, h \in V^4 - \{3\}$. Therefore, for job j in stage k, if $x_{ljh}^{k} = 1$, the value of this binary variable for all the other predecessors and machines must be zero. Thus, it is sufficient to consider only the variables that are equal to one when branching on the x-binary variables.

Briefly, instead of considering the following possible branches for the above example:

$$x_{l2h}^{4} = 1 \text{ or } x_{l2h}^{4} = 0, \forall l \in N^4 - \{2\}, h \in V^4$$

It is sufficient to consider:

$$x_{l2h}^{4} = 1, \forall l \in N^4 - \{2\}, h \in V^4$$

As a result, the number of branches is substantially reduced by branching on binary variables of the model (x-variables or Y-variables). Following the same procedure, the branching rule related to the B&P algorithm based on DWD2 is developed.

In order to show the impact of branching from the root node to the lower levels of the B&B tree, the amount of improvement on the lower bounds obtained by DWD3 for a small-, medium-, and large-size problem (demonstrated in Figures 21 through 23, respectively), are reported. The B&P optimization algorithm is capable of obtaining 1229.23, 6069.80, and 28391.02 as the lower bounds in the third, second, and second level of the small-, medium-, and large-size problem, respectively. Therefore, as a result of implementing branching from the root node of the B&B tree, an improvement of 3.73%, 6.42%, and 5.15% is obtained for the small-, medium-, and large-size problem, respectively.

6. EXPERIMENTAL SETUP AND DATA GENERATION

Developing an efficient approach, namely batch scheduling, to deal with the HFS scheduling problem along with proposing a robust algorithm to solve industry-size problems is the main purpose of this research. In order to show the benefits of the batching phase in group scheduling as well as to precisely evaluate the proposed algorithms in Section 4, three basic questions must be answered:

1) What is the benefit(s) of integrating the batching decision into the group scheduling approach?
2) Which algorithm (a basic or hybrid meta-heuristic), if any, outperforms the others for accepted standard benchmark of real problems and/or a given test problem?
3) How well do developed algorithms perform with respect to the optimal solution, or equivalently a tight lower bound as described in Section 5?

With respect to the first question, a comparison between the batch scheduling and group scheduling approaches show the benefits of implementing batch scheduling instead of group scheduling. To answer the second question, the most widely used benchmark for flow shop scheduling is that of Taillard (1993). Vallada et al. (2015) proposed a new benchmark of hard instances for the permutation flow shop scheduling problem with the objective of minimizing the makespan. But there is not any accepted standard benchmark of real problems or their representatives, for hybrid flow shop group scheduling problems. Therefore, in order to reflect the real-world instances, a comprehensive data generation mechanism designs the different parameters applied in the model. To answer the third question, the results of the developed algorithms are compared with optimal solutions or lower bounds obtained from CPLEX as well as B&P.

Since the size of a problem has a direct impact on the performance of any of the proposed meta-heuristic algorithms as well as the B&P algorithm, any comparison is performed on a set of randomly generated sample problems with the help of a comprehensive data generation mechanism. The notation of the MILP1 model is used to show the equations of the data generation mechanism. The parameters are developed as follows:

Problem size: the size of a problem is determined in terms of the problem *structure*, which has four different levels: (*small*, *small*), (*small*, *large*), (*large*, *small*) and (*large*, *large*). The first and second terms in the parenthesis denote the range of the number of groups and jobs within each group, respectively. Referring to Schaller et al. (2000) and some justifications for batch scheduling problems, "small" and "large" parameters in each level refer to a number generated from uniform distributions $unif[3, 5]$ and $unif[6, 10]$, respectively. It is worth noting that (small, small) and (large, large) levels can be considered as small- and large-size problems, respectively, while the other two levels can be considered as medium-size problems.

The range of the number of groups is in agreement with Schaller et al. (2000), while the range of the number of jobs within each group is unique to this research.

Machine capability and eligibility: Logendran and Subur (2004), Mehravaran and Logendran (2011), Pandya and Logendran (2010) considered three different types of machines based on the capability of processing: least, medium and most capable machines, which are eligible to process 50%, 70% and 85% of all jobs, respectively. In order to assign different capabilities to machines in k^{th} stage of HFS, if the number of machines is $v^k \leq 3$, then one of them is assigned to most capability and for the rest of them, $(v^k - 1)$ random numbers are generated in [0, 1]. The numbers with a value less than 1/3 and more than 1/3 will be counted separately. The largest and smallest count will be the numbers of least and medium capable machines, respectively; Otherwise, if the number of machines is $v^k \geq 3$, three of them are assigned to each capability. Thereafter, for the rest of them, $(v^k - 3)$ random numbers are generated in [0, 1]. The numbers with a value less than 1/3, between 1/3 and 2/3, and more than 2/3 will be counted separately. The largest and smallest count will be the extra numbers of least and most capable machines, respectively, and the remaining count will be the extra number of medium capable machines.

Job run time and machine setup time: the run times related to each stage are generated from $unif[\vartheta_i + 1, \vartheta_i + 20]$, where $\vartheta_i \; \forall i = 1, 2, 3$ are considered for the least, medium and most capable machines, respectively, and also ϑ_i is generated uniformly in [1, 10]. The largest and smallest random values are assigned to the least and most capable machines and the remaining one is assigned to the medium capable machine. The difficulty of flow line scheduling problems with sequence-dependent setup times depend on the balance between the average setup time and the average run time (Schaller et al. 2000). With respect to an adjustment and some justification for batch scheduling problems, *setup-to-runtime ratios* of 1.5:1, 3.5:1 and 5:1 have been considered for least, medium, and most capable machines, respectively. Subsequently, with respect to different ratios, the setup times related to each stage are generated from $unif[1, 30]$, $unif[1, 70]$, and $unif[1, 100]$, for least, medium, and most capable machines, respectively.

Scenario: the different combinations of the weights associated with the producer and customers in the objective function, i.e. α and β, are determined by different scenarios. Three levels are considered for this factor including $(\alpha = 0.4, \beta = 0.6)$, $(\alpha = 0.5, \beta = 0.5)$, and $(\alpha = 0.6, \beta = 0.4)$.

Machine availability time: the memoryless property of the Exponential distribution can simulate the machine availability time in terms of the average processing time of a group in each stage. Assume $\bar{S}^k$ and $\bar{t}^k$ represent the average sequence-independent setup time and the average run time of any group on all machines in k^{th} stage, respectively. Then, the machine availability time is determined as follows:

$$a_h^k = \exp\left(20 + \sum_{t=0}^{k-1} \bar{T}^t\right) \mid (\bar{T}^0 = 0) \tag{6.1}$$

$$\bar{T}^k = (\bar{S}^k + \bar{t}^k) \tag{6.2}$$

$$\bar{S}^k = \left(\left(\sum_{h\in V^k} \sum_{p\in I^k+\{0\}} \sum_{\substack{i\in I^k \\ i\neq p}} S_{pih}^k\right) \Big/ (g^2 \times v^k)\right) \tag{6.3}$$

$$\bar{t}^k = \left(\left(\sum_{i\in I^k} \sum_{j\in J_i^k} \sum_{h\in V_{ij}^k} t_{ijh}^k\right) \Big/ \left(\sum_{i\in I^k} \sum_{h\in V^k} \partial_{ih}^k\right)\right) \tag{6.4}$$

Job due date: the proper due dates should not be generated simply by picking numbers from a given distribution. Previous works (Kim et al. 2002, Pandya and Logendran 2010) showed that the generation of meaningful due dates in scheduling problems can positively affect the performance of the algorithms. Tardiness factor (τ) and due date range factor (R) can be used to define due dates of a problem instance. The tardiness factor, τ, is defined as $\tau = 1 - \bar{d}/C_{max}$ where $\bar{d}$ and C_{max} are the average due date and the maximum completion time of all jobs, respectively. The measure of variability of due dates, R, is defined as $R = (d_{max} - d_{min})/C_{max}$ where d_{max} and d_{min} are the maximum and minimum due date, respectively. Different combinations of τ and R from a uniform distribution, which are shown in Table 16, generate meaningful due dates with various characteristics (Suresh and Chaudhuri 1994). In other words, a large and small values of τ indicate tight and loose due dates (*due date tightness*), respectively. A wide range of due dates is determined by a large R, while a small R determines a narrow range of due dates.

Table 16. The range of τ and R

τ	0.2	0.2	0.2	0.5	0.5	0.5	0.8	0.8	0.8
R	0.2	0.5	0.8	0.2	0.5	0.8	0.2	0.5	0.8
Degree of tightness	Loose	Loose	Loose	Medium	Medium	Medium	Tight	Tight	Tight
Width of range	Narrow	Medium	Wide	Narrow	Medium	Wide	Narrow	Medium	Wide

R and τ have to be used simultaneously to determine due dates of jobs. In this research, the range factor has been set as $R = 0.2$, which provides narrow range of due dates. A random number from 0 to 1 is selected to generate a due date. If the random number falls into $[0, \tau]$, the due date is generated from a uniform distribution $[\bar{d} - R\bar{d},\ \bar{d}]$; and if the random number falls into $[\tau,\ 1]$, the due date is generated from a uniform distribution $[\bar{d},\ \bar{d} + (C_{max} - \bar{d})R]$ (Kim et al. 2002, Logendran and Subur 2004, Pandya and Logendran 2010). With the help of the following iterative equations, the completion time of any job j of group i in k^{th} stage (x_{ij}^{k}) and, consequently, C_{max} is estimated.

$$\begin{cases} x_{i1}^{k} = \left(\left(\sum_{h\in V_{i1}^{k}} \left(\max\left(x_{i1}^{k-1}, mr_{i}^{k} + \gamma \bar{s}_{i}^{k}\right) + \left(t_{i1h}^{k}\right)\right)\right) \Big/ \xi_{i1}^{k} \right) \\ x_{ij}^{k} = \max\left(x_{ij}^{k-1}, x_{i(j-1)}^{k}\right) + \left(\left(\sum_{h\in V_{ij}^{k}} t_{ijh}^{k} \right) \Big/ \xi_{ij}^{k} \right) \end{cases} \tag{6.5}$$

where $\bar{s}_{i}^{k}$ is the average run time of group i in k^{th} stage. mr_{i}^{k}, the first available time of machines to start processing group i in k^{th} stage, is estimated as follows:

$$mr_{i}^{k} = \begin{cases} \min\left(x_{p(n_p)}^{k}\right), & p = (i-1), (i-2), \dots, \left(i - v^{k}\right) \\ a_{i}^{k} \qquad\qquad , & \text{otherwise} \end{cases} \tag{6.6}$$

where $x_{p(n_p)}^{k}$ is the completion time of the last job in group p in k^{th} stage.

γ was proposed by Logendran et al. (2007) as an adjustment on the average setup time because of sensitivity of the completion time to the sequence-dependent family setup times. They suggested a linear relationship between γ and CV, where CV is the coefficient of variation related to the sequence-dependent setup times. This relationship can be described by the interpolation of $(CV, \gamma) = (0.01, 0.9)$ and $(CV, \gamma) = (1, 0.1)$. Finally, $x_{ij}^{0} = r_{ij}$ in the first stage and $C_{max} = \max_{\forall i \in G, j \in G_i} x_{ij}^{m}$, in the last stage.

Desired lower bounds on batch sizes: the proper lower bounds on batch sizes are generated by balancing the setup time and cumulative run time of a batch. An estimation of the minimum number of jobs assigned to a batch of a group on each machine is determined by initial ratio developed for each group on each machine in k^{th} stage (IR_{ih}^{k}) along with pre-determined base ratio (BR) and adjusted ratio ($AR = k \times IR_{ih}^{k}$) developed for each problem structure. Initial ratio of a group is considered as the average setup time to the average run time. Base ratio is equal to 1.3, 5, 7, and 8 for (*small*, *small*), (*small*, *large*), (*large*, *small*) and (*large, large*) levels, respectively. coefficient k is equal to 0.3, 0.1, 0.05, and 0.04 for (*small*, *small*), (*small*, *large*), (*large*, *small*) and (*large*, *large*) levels, respectively.

$$IR_{ih}^{k} = \bar{s}_{ih}^{k} \Big/ \bar{t}_{ih}^{k} \tag{6.7}$$

$$CR^{B}{}_{ih}^{k} = \left(\bar{s}_{ih}^{k} + B\bar{t}_{ih}^{k}\right) \Big/ \left(B\bar{t}_{ih}^{k}\right) \tag{6.8}$$

$$MR_{ih(B\to B+1)}^{k} = \left(CR^{B}{}_{ih}^{k}\right) \Big/ \left(CR^{(B+1)}{}_{ih}^{k}\right), \qquad \forall\, B = 1,2,\dots,n_i - 1 \tag{6.9}$$

$$TV_{ih}^{k} = 1 + \left(\bar{s}_{ih}^{k} \Big/ \left(AR\left(\bar{s}_{ih}^{k} + \bar{t}_{ih}^{k}(AR+1)\right)\right)\right) \tag{6.10}$$

where $\bar{s}_{ih}^{k}/\bar{t}_{ih}^{k}$ is the average setup time/ job run time of group i on machine h in k^{th} stage, which is obtained by Eq. (4.60)/Eq. (4.64). B is the number of jobs assigned to a batch. And finally, $CR^{B}{}_{ih}^{k}$, $MR_{ih(B\to B+1)}^{k}$, and TV_{ih}^{k} are cumulative ratio, movement ratio, and threshold value of group i on machine h in k^{th} stage, respectively. Then, LB_{ih}^{k} is determined from one of the conditions shown in Table 17.

Table 17. Pseudo-code for determining the desired lower bounds on batch sizes

```
if $IR_{ih}^{k} \leq BR$ then
    $LB_{ih}^{k} = 1$
else if $IR_{ih}^{k} > BR$ & $AR \leq 1$ then
    $LB_{i}^{h} = 2$
else if $AR > 1$ then
    $LB_{ih}^{k} = (B+1) \mid \min_{B\in\{1,2,\dots,n_i-1\}} \left(\left|MR_{i(B\to B+1)}^{h} - TV_{ih}^{k}\right|\right)$
end if
```

Other parameters: the number of stages of hybrid flow shop and the number of machines in each stage are generated from $unif[2,7]$ and $unif[1,6]$, respectively. Apart from this, the job release times and job weights are generated based upon an Exponential distribution with a mean of 20, i.e., $exp(20)$, and uniform distribution, i.e., $unif[1,4]$, respectively.

7. RESULTS

In this section, first two approaches to deal with the scheduling problem addressed in this research are compared to show the benefit(s) of batching: group scheduling and batch scheduling. Then, the computational experiments are performed to compare the performance of the proposed meta-heuristic algorithms. Finally, the performance of the proposed algorithms is evaluated with respect to optimal solutions or tight lower bounds obtained by CPLEX and proposed B&P algorithm. Also, a comparison between implemented DWDs based on column generation determines the performance of each of the DWD decomposition techniques. For this purpose, the proposed search algorithms are implemented using C# programming language, while the optimal/upper bounds and lower bounds are obtained from solving the MILP models or SPs with CPLEX 12.2 (CPLEX 2009). All runs relating to parameter tuning and experiments have been performed on identical computers with Intel(R) Core(TM) 2 Duo CPU T9300 @ 2.50 GHz processors & 4.00 GB of RAM.

7.1. Batch scheduling vs. group scheduling

Group scheduling developed by Bozorgirad and Logendran (2013) is arguably the best available approach to deal with the research problem addressed here, specifically with respect to comparing the performances of both. It is therefore important to compare the performance of the proposed batch scheduling approach with this approach to see the benefits of batching. With respect to three days' time limit ($CT_{limit} = 3d$), several experiments (19 problems shown in Table 18) conducted on the benchmark problem instances confirm the benefits of batching.

The following results 1 through 3 are obtained from comparison between the group scheduling and batch scheduling approaches:

Result 1. The benefits of batching are especially obvious for instances with more machines per stage (on average). This confirms that batch scheduling can utilize the available machine capacities and perform timely processing of jobs with higher priorities and, consequently, increase scheduling flexibility.

Result 2. Although dividing groups into batches promotes more setups, they may be performed during machine idle times (Shen et al. 2014). More importantly, forming different combinations of batch compositions related to stages allows varied job assignment and sequences on each machine, which can effectively reduce total processing time of jobs. Consequently, it may not negatively affect the completion time of all jobs and their tardiness.

Result 3. Although there is a trade-off between any changes in jobs' completion times, jobs' tardiness, and setup times, batch scheduling gives a schedule, which is at least as good as the optimal schedule obtained by group scheduling.

Table 18. Performance of batch scheduling vs. group scheduling

Test problem	# of stages	# of groups	# of jobs	# of machines	Group Scheduling (GS) Obj Func Val.	Group Scheduling (GS) Comp Time (Sec)	Batch Scheduling (BS) Obj Func Val.	Batch Scheduling (BS) Comp Time (Sec)	Batch Scheduling (BS) Lower Bound	PImp BS vs GS
1	3	2	8	6	880.20	0.55	862.80	83.0		2.0%
2	3	2	8	12	1170.60	1.08	1008.80	138748.5		13.8%
3	2	4	8	2	1535.40	1.19	1535.40	6.8		0.0%
4	3	2	8	3	1720.08	1.42	1720.08	9.1		0.0%
5	3	3	9	6	2215.80	1.36	2006.82	24873.4		9.4%
6	7	4	12	28	2733.20	351.00	2235.68	Fixed	1897.45	18.2%
7	7	4	12	28	3764.60	10.09	3058.26	39398.5		18.8%
8	7	3	12	28	3299.40	7.21	2981.61	29847.9		9.6%
9	4	5	20	4	8257.00	337024.00	8189.00	Fixed	7199.34	0.8%
10	2	4	16	2	3436.40	14.00	3420.70	245598.1		0.5%
11	2	5	15	10	1447.00	11.37	1244.42	Fixed	1038.78	14.0%
12	3	5	20	15	2148.00	429203.00	1808.40	Fixed	1692.82	15.8%
13	2	2	10	4	1860.00	2.00	1699.67	30938.7		8.6%
14	5	2	14	25	3373.60	5235.00	2668.85	Fixed	2482.70	20.9%
15	2	6	24	8	3355.20	447923.00	2869.03	Fixed	2490.28	14.5%
16	2	7	21	10	1928.40	-	1679.05	Fixed	1423.51	12.9%
17	4	5	15	24	2045.00	-	1649.49	Fixed	1481.19	19.3%
	Average				2657.05	81319.08	2390.47	151947.29	2463.26	10.5%

The sign "-" indicates that no computational time is reported by group scheduling.

Thus, from the benchmark problem instances shown in Table 18, it is beneficial to drop the GTAs. By applying the proposed batch scheduling instead of group scheduling, further improvement in the objective function value can be obtained (from 0.5% up to approximately 21%). Using the following one-way hypothesis test at a 5% significance level, the result obtained from a paired *t*-test on the average objective function values of both approaches showed there is a statistically significant reduction in the objective function value of group scheduling by allowing for the possibility of splitting groups as multiple batches ($P_{value} < 0.00001$).

$$\begin{cases} H_0: \mu_{BS} - \mu_{GS} = 0 \\ H_a: \mu_{BS} - \mu_{GS} < 0 \end{cases}$$

7.2. Best algorithm

The basic TS and PSO along with TS/PR PSO/LSA as hybrid algorithms developed in Section 4 must be compared to each other to determine what type of meta-heuristic(s) has better performance in dealing with batch scheduling problems in HFS. In this case, a comprehensive set of experiments have been designed based upon a split-plot design (Montgomery 2009) with five factors, shown in Table 19. The main-plot factors include the *structure* (*Str*) of the problem, *setup to run time ratio* (*StoR*), *due date tightness* (*DDT*), and *scenario* (*Sc*), while the eight different *algorithms* (*Alg*) belong to the sub-plot factor.

Table 19. Factors and their levels in the experiment

Factor name		Levels
Whole-plot		
Structure	*Str*	(Small, Small), (Small, Large), (Large, Small), (Large, Large)
Setup-to-run time ratio	*StoR*	1.5, 3.5, 5
Due date tightness	*DDT*	$tight$ (0.2), $Medium$ (0.5), $Loose$ (0.8)
Scenario	*Sc*	$(\alpha = 0.4, \beta = 0.6)$, $(\alpha = 0.5, \beta = 0.5)$, $(\alpha = 0.6, \beta = 0.4)$
Sub-plot		
Algorithm	*Alg*	$Alg1, Alg2, Alg3, Alg4, Alg5, Alg6, Alg7, Alg8$

Eight levels have been considered for *Alg* factor to balance between path construction techniques and local search algorithms for TS/PR and PSO/LSA algorithms, respectively. Constructing a good path between two elite solutions in the population is a key step in TS/PR. Therefore, based on the importance of introducing the promising solutions in relinking paths, four algorithms are generated as follows:

Alg1 *PR is excluded and a basic TS algorithm is developed*
Alg2 *PR explores trajectories connecting elite solutions based on only LCS-based construction*
Alg3 *PR explores trajectories connecting elite solutions based on a combination of LCS- & block-based constructions*
Alg4 *PR explores trajectories connecting elite solutions based on a combination of LCS- & swap-based contractions*

Although implementation of LSA for all particles within a population may take a considerable amount of time, there should be a trade-off in the frequency of using LSA within the structure of PSO. Therefore, based on the frequency of implementing LSA to improve the performance of the PSO, four algorithms are generated as follows:

Alg5 *LSA is excluded and PSO that stands for the basic PSO algorithm is developed*
Alg6 *LSA is implemented for only* ${P_{best}}^{Global}$ *vector at each iteration*
Alg7 *LSA is implemented for* P_{best} *vectors of particles at each iteration*
Alg8 *LSA is implemented for all particles within the population at each iteration*

The ten replications, each with a different number of groups, jobs belonging to groups, stages, and machines belonging to each stage have randomly been generated for any combination of *Str*, *StoR*, *DDT*, and *Sc* factors. Each of these replications has been solved by all eight algorithms, which resulted in a total number of 8640 computer runs (108 combination of *Str*, *StoR*, *DDT*, and *Sc* × 10 replications × 8 *Alg* = 8640). The statistical model for this design is:

$$y_{ijklmn} = \mu + \tau_i + \vartheta_j + \rho_k + \gamma_l + \delta_m + (\vartheta\rho)_{jk} + (\vartheta\gamma)_{jl} + (\vartheta\delta)_{jm} + (\rho\gamma)_{kl} + (\rho\delta)_{km} + (\gamma\delta)_{lm} + (\vartheta\rho\gamma)_{jkl} + (\vartheta\rho\delta)_{jkm} + (\rho\gamma\delta)_{klm} + (\vartheta\rho\gamma\delta)_{jklm} + \theta_{ijklm} + \varphi_n + (\vartheta\varphi)_{jn} + (\rho\varphi)_{kn} + (\gamma\varphi)_{ln} + (\delta\varphi)_{mn} + (\vartheta\rho\varphi)_{jkn} + (\vartheta\gamma\varphi)_{jln} + (\vartheta\delta\varphi)_{jmn} + (\rho\gamma\varphi)_{kln} + (\rho\delta\varphi)_{kmn} + (\gamma\delta\varphi)_{lmn} + (\vartheta\rho\gamma\varphi)_{jkln} + (\vartheta\rho\delta\varphi)_{jkmn} + (\rho\gamma\delta\varphi)_{klmn} + (\vartheta\rho\gamma\delta\varphi)_{jklmn}$$

$i = 1,2, \dots ,10$; $j = 1,2,3,4$; $k, l, m = 1, 2, 3$; and $n = 1, 2, \dots , 8$ where μ is the overall mean effect, τ_i is the replicate effect, ϑ_j is the effect of j^{th} level of *Str*, ρ_k is the effect of k^{th} level of *StoR*, γ_l is the effect of l^{th} level of *DDT*, δ_m is the effect of m^{th} level of *Sc*, and finally φ_n shows the effect of n^{th} level of *Alg*. The resulting ANOVA table is presented in Table 20.

Table 20. ANOVA table on the natural log of data

	Sum of Square	***Degree of Freedom***	***Mean of Square***	***F-Statistic***	***P-value***
Source (main plot)					
Rep (or Blocks)	17.943	9	1.994	0.890	0.533
Str	4,363.881	3	1,454.627	649.438	0.000
StoR	412.998	2	206.499	92.194	0.000
DDT	164.358	2	82.179	36.690	0.000
Sc	91.300	2	45.650	20.381	0.000
Str:StoR	3.045	6	0.508	0.227	0.968
Str:DDT	20.623	6	3.437	1.535	0.164
StoR:DDT	2.295	4	0.574	0.256	0.906
Str:Sc	8.236	6	1.373	0.613	0.720
StoR:Sc	9.185	4	2.296	1.025	0.393
DDT:Sc	38.646	4	9.662	4.314	0.002
Str:StoR:DDT	32.713	12	2.726	1.217	0.266
Str:StoR:Sc	29.272	12	2.439	1.089	0.366
Str:DDT:Sc	15.695	12	1.308	0.584	0.856
StoR:DDT:Sc	24.017	8	3.002	1.340	0.219
Str:StoR:DDT:Sc	35.595	24	1.483	0.662	0.890
Main Plot Error	2,156.951	963	2.240		
Source (Sub-plot)					
Alg	3.623	7	0.518	193.490	0.000
Str:Alg	0.582	21	0.028	10.355	0.000
StoR:Alg	0.230	14	0.016	6.152	0.000
DDT:Alg	0.000	14	0.000	0.000	1.000
Sc:Alg	0.115	14	0.008	3.073	0.000
Str:StoR:Alg	0.173	42	0.004	1.540	0.014
Str:DDT:Alg	0.087	42	0.002	0.773	0.854
StoR:DDT:Alg	0.107	28	0.004	1.429	0.067
Str:Sc:Alg	0.116	42	0.003	1.035	0.409
StoR:Sc:Alg	0.000	28	0.000	0.000	1.000
DDT:Sc:Alg	0.113	28	0.004	1.514	0.040
Str:StoR:DDT:Alg	0.220	84	0.003	0.978	0.536
Str:StoR:Sc:Alg	0.288	84	0.003	1.282	0.043
Str:DDT:Sc:Alg	0.159	84	0.002	0.708	0.980
StoR:DDT:Sc:Alg	0.217	56	0.004	1.446	0.017
Str:StoR:DDT:Sc:Alg	0.475	168	0.003	1.056	0.298
Subplot Error	18.201	6804	0.003		
Total	7,451.459	8639			

Rep=Replicate;Str=Structure;StoR=Setup to Runtime Ratio;DDT= Due Date Tightness;Sc=Scenario; Alg=Algorithm

Due to violations in normality assumption and constant variance assumption, the natural logarithm transformation of the response variable has been employed to totally resolve all the deviations (R 2. 13.0., 2011). Based on the ANOVA table for these experiments, all factors in the main-plot (*Str*, *StoR*, *DDT*, and *Sc*) impose statistically significant effects on the objective function of the test problems ($P_{value} < 0.00001$), while the effect of their interactions is not significant except the interaction between *DDT:Sc*. After accounting for the effect of those factors, the ANOVA table reveals convincing evidence of non-zero differences between levels of algorithms ($P_{value} < 0.00001$). With the help of Tukey test on the levels of *Alg* factor, *Alg8* has the best performance among the eight algorithms with a confidence level of 95%.

The deviation of all algorithms from the best algorithm is calculated as $dev = ((Alg_i - Alg_{best})/(Alg_{best}))\times 10, \forall i = 1, 2, \dots, 7$ to demonstrate the relative performance of all algorithms with each other. These deviations are depicted in the left side of Figure 24 by a box plot diagram. The results of relative performances indicate *Alg4* and *Alg7* have the second and third best performance with the mean deviations of 1.5% and 3.0%, respectively. Since there is no meaningful difference between the performances of *Alg2*, *Alg3*, and *Alg6*, they are considered as the fourth best algorithms with the mean deviations of 4.7%, 4.6%, and 4.2%, respectively. Finally, *Alg1* and *Alg5* are demonstrated as the algorithms with the largest deviations around 6.7% and 8.8%, respectively. The 95% confidence intervals (CI) for the means of these deviations, depicted in the right side of Figure 24, suggest no significant difference between the performance of *Alg4* and *Alg8* because of the smallest deviation from *Alg8* along with the short range of deviations. Apart from this, the box plot of *Alg4* depicted in Figure 24 suggests *Alg4* can give a better solution compared to *Alg8* in some problems.

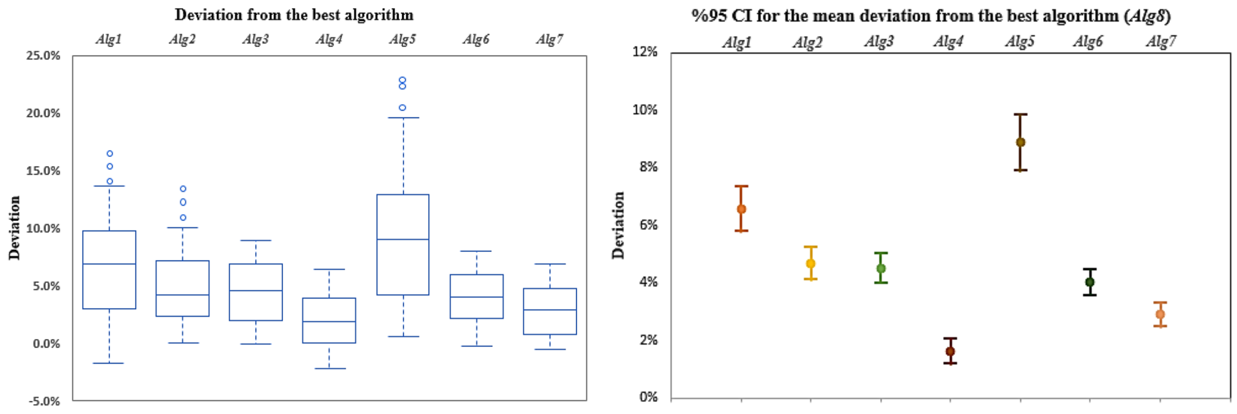


Figure 24. Deviations from the best algorithm

The following results 4 through 6 are obtained from comparison between algorithms:

Result 4. Since a basic local search algorithm can decompose the batch scheduling problem into several hierarchical levels to take advantage of batching, it presents a better performance compared to a basic population-based algorithm, especially when there is an exhaustive combination enumeration between batch compositions of all groups in all stages of a problem. This being the case, *Alg1* has the ability to generate solutions with better quality compared to *Alg5*.

Result 5. Due to involving several moves at the same time for creating a neighbor solution, a population-based structure is able to find good quality solutions in less iterations compared to local search algorithms for batch scheduling in HFS. In addition, a population-based algorithm enhanced with a local search structure is able to integrate the batching phase into the scheduling phase. Therefore, *Alg8* is able to identify good quality solutions compared to *Alg4*, in less computational time.

Result 6. Based on a test problem in (large, large) level, a comparison of the iterative improvement in the objective function value between TS/PR and PSO/LSA, depicted in Figure 25, indicates PSO/LSA is able to obtain the same best solution compared to TS/PR in less iterations and, subsequently, in less computational time. Although TS/PR and PSO/LSA give the same best solution, TS/PR gradually improves the objective function in each stage, while PSO/LSA has more and less improvement in the objective function in the initial and final iterations, respectively, in each stage. As a result, PSO/LSA converges to the best solution faster than TS/PR. Since the stage-based interdependency strategy is implemented for each algorithm, there are unusual increases in the objective function value when the algorithm goes through stages.

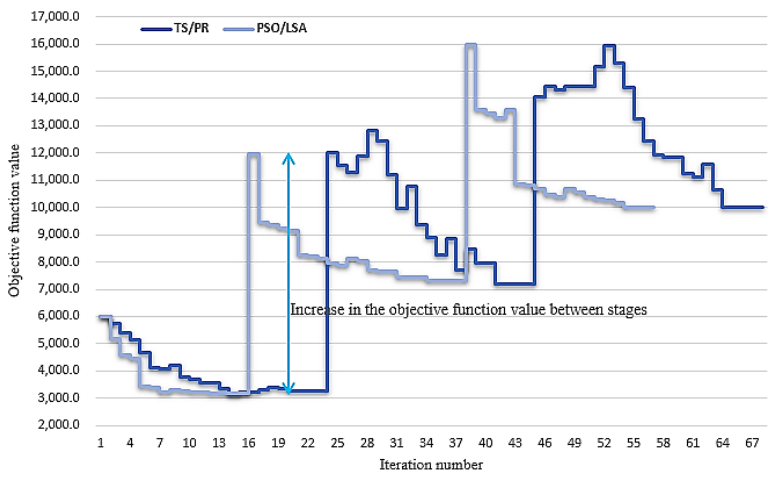


Figure 25. Comparison between iterative improvement of TS/PR and PSO/LSA

There are strong evidences of a nonzero difference between the interaction effect of *Str:Alg*, *StoR:Alg*, and *Sc:Alg*. Since the interpretation of a high rank interaction is quite difficult to explain, only the interactions between *Str:Alg*, *StoR:Alg*, and *Sc:Alg* are explained with the help of Tukey test on each level of *Str*, *StoR*, and *Sc* factors to compare the performance of the developed algorithms. Based on P_{value} at a significance level of 5% for each comparison in each level of *Str*, *StoR*, and *Sc*, it can be concluded that the problem structures, different ratios between setup-to-run time, and different importance coefficients associated with a producer and customers (scenarios) have an effect on the performance of the developed algorithms.

Generally, since the average objective function value of *Alg8* is less than all the other developed algorithms for all levels of aforementioned factors, it can be concluded that *Alg8* provides better solutions for the proposed research problem, particularly for problems with (large, large) structure. Although *Alg4* presents a close performance compared to *Alg8* for problems with (small, large) & (large, small) structures, *Alg8* presents a slightly better performance compared to *Alg4* for problems with (large, large) structure. With respect to the interaction effect of *Str:Alg*, it can be concluded that the total number of jobs of all groups is very sensitive to the processing time and tardiness of jobs. In relation to the interaction effect of *Sc:Alg*, the difference between the developed algorithms, particularly *Alg8* and *Alg4*, is more pronounced when customers' satisfaction has high priority compared to the production cost, i.e., a larger coefficient assigned to customers.

Finally, since a bi-criteria objective is considered in the research problem, not only the interaction of the developed algorithms and *Sc* is pronounced, but also the individual contribution of *Sc* as a factor is equally pronounced. On the other hand, the ANOVA table reveals moderate evidence of non-zero differences between different levels of interaction between *DDT:Sc*. This being the case, a paired *t*-test is performed to compare the different levels of *Sc* with respect to the average objective function values of the two best algorithms, i.e., *Alg4* and *Alg8*, i.e., $1\times3 = 3$ comparisions. Likewise, another paired *t*-test is performed to compare the different levels of *DDT:Sc*, i.e., $3\times3 = 9$ comparisions. Sc_1, Sc_2, and Sc_3 are referred to $(\alpha = 0.4, \beta = 0.6)$, $(\alpha = 0.5, \beta = 0.5)$, and $(\alpha = 0.6, \beta = 0.4)$ scenario, respectively. The principal results of a paired *t*-test performed to compare the different levels of *Sc* are shown in Table 21.

Table 21. Paired *t*-test for scenarios

	Paired differences							
	Mean	Std. deviation	Std. error mean	95% confidence interval of the difference		***t***	***df***	**Sig. (2-tailed)**
Pair				**Lower**	**Upper**			
Sc_1 - Sc_2	814.424	861.375	157.265	495.176	1275.209	5.179	35	**9.34384E-06**
Sc_1 - Sc_3	730.967	972.416	177.538	370.565	1251.154	4.117	35	**2.22097E-04**
Sc_2 - Sc_3	83.456	1129.523	206.222	-335.174	687.686	0.405	35	**6.88167E-01**

Based on P_{value} at a 5% significance level for each comparison, it can be concluded that the objective function value is sensitive to the scenario. In other words, the average objective function value obtained from Sc_1 is considerably less than the ones obtained from Sc_2 and Sc_3. In addition, although the average objective function value obtained from *Sc*$_2$ is less than the one obtained from *Sc*$_3$, there is no statistical evidence of a difference between *Sc*$_2$ and *Sc*$_3$.

The principal results of a paired *t*-test performed to compare the different levels of *Sc* for each level of due date tightness are shown in Table 22. Based on P_{value} at a 5% significance level for each comparison in each due date tightness level, it can be concluded that there is a significant difference between the results of these scenarios in each level of due date tightness. In other words, since the average objective function value obtained from *Sc*$_1$ is considerably less than the ones obtained from *Sc*$_2$ and *Sc*$_3$ for all due date tightness levels, it can be concluded that *Sc*$_1$ reduces the objective function value of a problem more in comparison to when *Sc*$_2$ and *Sc*$_3$ are considered, particularly when the jobs have tight and moderate due dates, i.e., $DDT = 0.2$ or $DDT = 0.5$. In addition, *Sc*$_2$ reduces the objective function value of a problem slightly more than *Sc*$_3$ when the jobs have moderate due date. With respect to the interaction effect of *DDT:Sc*, it can be concluded that the objective function value is simultaneously very sensitive to the importance coefficients associated with the producer and customers along with the degree of tightness of jobs' due date. These results are in agreement with the results obtained for scenario sensitivity analysis.

Table 22. Paired *t*-test for interaction between scenarios and due date tightness

		Paired differences							
		Mean	Std. deviation	Std. error mean	95% confidence interval of the difference		*t*	*df*	Sig. (2-tailed)
DDT	Pair				Lower	Upper			
Tight	*Sc*$_1$ - *Sc*$_2$	816.244	526.428	96.112	604.797	1027.690	8.493	11	3.68457E-06
	Sc$_1$ - *Sc*$_3$	795.970	734.903	134.174	500.787	1091.153	5.932	11	9.83980E-05
	Sc$_2$ - *Sc*$_3$	20.274	962.781	175.779	-366.440	406.987	0.115	11	9.10258E-01
Medium	*Sc*$_1$ - *Sc*$_2$	342.061	224.474	40.983	251.898	432.224	8.346	11	4.35568E-06
	Sc$_1$ - *Sc*$_3$	606.870	965.104	176.203	219.224	994.517	3.444	11	5.48432E-03
	Sc$_2$ - *Sc*$_3$	264.810	960.946	175.444	-121.167	650.786	1.509	11	1.59381E-01
Loose	*Sc*$_1$ - *Sc*$_2$	567.442	419.725	76.631	398.854	736.030	7.405	11	1.35159E-05
	Sc$_1$ - *Sc*$_3$	790.062	1229.625	224.498	296.166	1283.957	3.519	11	4.80488E-03
	Sc$_2$ - *Sc*$_3$	222.619	1427.341	260.596	-350.691	795.930	0.854	11	4.11173E-01

In conclusion, since both objectives (total completion time and total tardiness of jobs) are not in conflict, the tardiness of a job is either reduced or not changed when its completion time reduces. In addition, the completion time of a job is certainly reduced when its tardiness (positive value of lateness) reduces. Regarding the two previous statements, the objective function value is reduced more when larger importance coefficient is assigned to customers and/or tight due date is considered for the jobs.

7.3. Algorithms versus optimal solutions and lower bounds

One of the main purposes of this research is to take the benefits of integrating the batching decision into the group scheduling approach. Therefore, the performance of batch scheduling and group scheduling should be compared. Another purpose of this research is to develop robust meta-heuristic algorithms to deal with the batch scheduling problem in HFS. Therefore, the performance of the meta-heuristic algorithms should be evaluated with respect to either optimal solutions or tight lower bounds obtained from the B&P algorithm. For these comparisons and evaluations, a total of 30 sample problems, generated by the comprehensive data generation mechanism (Section 6) and divided into three groups of 10 problems, are tested. Those three groups represent (small, small), (small, large) & (large, small), and (large, large) problem structures. Observe that, as noted before, (small, large) & (large, small) problems are considered as one level or medium-size problems.

Given the computational time limit (the common 8-hour work-shifts), the benefits of integrating the batching decision into the group scheduling approach are again demonstrated by comparing the results of upper bounds/optimal solutions of batch scheduling compared to group scheduling. The MILP3 model is implemented for group scheduling with regard to $LB_{ih}^{k} = n_i, \ \forall i \in I^k; h \in V^k; k \in K$ and some justifications. Apart from this, the performance of the best meta-heuristic algorithm is evaluated against optimal solutions as well as lower bounds obtained from CPLEX and the B&P algorithm. Eight-hour and one-hour time limits are considered for CPLEX / the B&P algorithm and meta-heuristic algorithms, respectively. The results are shown in Tables 23 through 25 for (small, small), (small, large) & (large, small), and (large, large) levels, respectively. Each test problem is represented as a problem class (i.e., *level| # of groups| # of all jobs| # of stages| # of all machines*). The best upper bounds obtained by the best TS/PR and PSO/LSA algorithms (i.e., *Alg4* and *Alg8*) are reported under the UB_{Alg} column.

As discussed in Sections 4.1.5 and 5.3.1.3 and based on our preliminary experiments, the MILP3 model along with the RMILP model are considered to find optimal solutions or good quality lower bounds, particularly for small-size problems. In addition, DWD3 along with DWD2 decomposition techniques are considered in the B&P algorithm to develop tight lower bounds, particularly for medium- and large-size problems. By considering $LB_{(B\&P_{DWD2})}$ and $LB_{(B\&P_{DWD3})}$ as the lower bound obtained by the B&P algorithm with the help of DWD2 and DWD3, respectively, and some justification on the Pseudo-code in Table 1, the best lower bound is determined as follows:

Table 23. The benefits of batching, performance of UB_{Alg} vs UB_{BS}, and performance of UB_{Alg} and UB_{BS} vs LB_{BS} for (Small, Small) level of problems

Level	Test Problem	$Level\|g\|\sum_{i\in g} n_i\|m\|\sum_{k\in m} v^k$	Group Scheduling (GS)			Batch Scheduling (BS)											Average Deviation				
			MILP Model			RMILP Model				MILP3 Model			Meta-heuristic		B&P						
			UB_{GS}	LB_{GS}	CT_{GS}	UB_{BS}	LB_{BS}	CT_{BS}	LB^k_{ih}	UB_{BS}	LB_{BS}	CT_{BS}	UB_{Alg}	CT_{Alg}	$B\&P_{DWD2}$	$B\&P_{DWD3}$	UB_{GS} vs UB_{BS}	UB_{GS} vs UB_{Alg}	LB_{BS} vs UB_{BS}	LB_{BS} vs UB_{Alg}	UB_{BS} vs UB_{Alg}
(Small, Small)	1	$SS\|3\|9\|3\|5$	2209.16		16.73	**2209.16**		219.19	No	2209.16		29839.93	2209.16	198.06	2054.52	2076.61	0.0	0.0	0.0	0.0	0.0
	2	$SS\|5\|18\|6\|15$	2744.85		37.03	2473.51		1219.01	Yes	**2529.57**	2459.98	Fixed	2537.68	268.23	2402.48	**2478.05**	7.8	7.5	2.0	2.4	-0.3
	3	$SS\|5\|18\|6\|16$	4388.81		202633.70	**4159.71**		246294.06	No	4202.76	4038.67	Fixed	4178.08	21.88	4060.54	4108.11	5.2	4.8	0.0	0.4	-0.4
	4	$SS\|3\|9\|2\|3$	7369.70		54.06	**7008.44**		222.88	No	7008.44		28084.23	7098.44	119.12	6447.76	6447.76	4.9	3.7	0.0	1.3	-1.3
	5	$SS\|2\|10\|4\|7$	6434.77		61.98	6168.78		662.43	Yes	**6308.60**		48755.01	6368.6	456.24	4935.02	5803.91	2.0	1.0	0.0	1.0	-1.0
	6	$SS\|4\|12\|6\|13$	4247.74		87495.40	**3953.57**		214573.27	Yes	**3997.88**	3938.28	Fixed	4001.88	636.14	3281.46	3558.11	5.9	5.8	1.1	1.2	-0.1
	7	$SS\|4\|13\|2\|7$	4648.82		14.16	4236.73		46.40	Yes	**4284.21**		207586.9	4298.21	532.41	4024.89	4027.16	7.8	7.5	0.0	0.3	-0.3
	8	$SS\|5\|15\|4\|11$	6849.68		12.18	**5976.86**		758.20	No	6043.84	5893.02	Fixed	6002.45	597.24	5080.33	5137.26	12.7	12.4	0.0	0.4	-0.4
	9	$SS\|3\|11\|4\|7$	5029.62		36.93	**4635.14**		633.42	No	4635.14		159687	4685.48	349.27	3986.22	4264.33	7.8	6.8	0.0	1.1	-1.1
	10	$SS\|3\|12\|4\|9$	7872.20		43.66	**7013.59**		635.95	No	7254.77	6980.02	Fixed	7213.59	205.83	6592.77	6166.55	10.9	8.4	0.0	2.9	-2.9
																Average	**6.5**	**5.8**	**0.3**	**1.1**	**-0.8**

UB, *LB*, *CT* stand for upper bound, lower bound, and computational time, respectively. NK stands for "Not Known".
Bold numbers represent *UB* and *LB* selected for a test problem.
Under average deviation column, LB_{BS} is determined in terms of *LB* related to RMILP model, MILP3 model, and B&P algorithm.
Under average deviation column, UB_{BS} is determined in terms of *UB* related to RMILP and MILP3 models.
B&P algorithm applies DWD3 approach to decompose the problem.
UB_{GS} and LB_{GS} are determined based on the MILP3 model by relaxing desired bound constraints, i.e., $LB^k_{ih} = n_i$.

Table 24. The benefits of batching, performance of UB_{Alg} vs UB_{BS}, and performance of UB_{Alg} and UB_{BS} vs LB_{BS} for (Small, Large) & (Large, Small) levels of problems

Level	Test Problem	$Level\|g\|\sum_{i\in g} n_i\|m\|\sum_{k\in m} v^k$	Group Scheduling (GS)			Batch Scheduling (BS)											Average Deviation				
			MILP Model			RMILP Model				MILP3 Model			Meta-heuristic		B&P						
			UB_{GS}	LB_{GS}	CT_{GS}	UB_{BS}	LB_{BS}	CT_{BS}	LB_{ih}^k	UB_{BS}	LB_{BS}	CT_{BS}	UB_{Alg}	CT_{Alg}	$B\&P_{DWD2}$	$B\&P_{DWD3}$	UB_{GS} vs UB_{BS}	UB_{GS} vs UB_{Alg}	LB_{BS} vs UB_{BS}	LB_{BS} vs UB_{Alg}	UB_{BS} vs UB_{Alg}
(Small, Large) & (Large, Small)	1	$SL\|3\|19\|4\|14$	12233.35		243654.29	9203.07	8282.76	Fixed	NK	**10434.33**	5247.40	Fixed	10409.68	983.87	9082.69	**10289.02**	14.7	14.9	1.4	1.2	0.2
	2	$SL\|3\|24\|3\|8$	7637.09		244158.10	6812.02	5585.86	Fixed	NK	**6888.35**	3774.32	Fixed	6848.82	2109.40	6700.82	**6739.85**	9.8	10.3	2.2	1.6	0.6
	3	$SL\|4\|28\|3\|5$	13825.68		207876.19	**11651.24**		249271.29	Yes	**12334.67**	6380.52	Fixed	12024.82	2153.16	9204.48	11209.42	10.8	13.0	5.5	3.2	2.5
	4	$SL\|4\|31\|6\|21$	4977.98	4873.68	Fixed	4005.82	2884.19	Fixed	NK	**4250.71**	1651.68	Fixed	4182.98	1354.13	3929.04	**4129.20**	14.6	16.0	2.9	1.3	1.6
	5	$SL\|4\|25\|3\|14$	9817.66		190738.10	8746.86		250098.02	Yes	**9047.65**	4180.41	Fixed	9084.83	1219.61	7347.36	**8894.48**	7.8	7.5	1.7	2.1	-0.4
	6	$LS\|8\|55\|6\|9$	13612.69	11156.51	Fixed	11219.62	7517.15	Fixed	NK	**11877.74**	6175.13	Fixed	10984.24	3027.81	**9983.56**	9338.62	12.7	19.3	21.4	17.6	7.5
	7	$LS\|9\|54\|5\|12$	13276.16	11812.27	Fixed	12228.02	8926.45	Fixed	NK	**12365.05**	7183.75	Fixed	11746.91	1204.21	10338.88	**10837.46**	6.9	11.5	12.4	8.4	5.0
	8	$LS\|8\|59\|2\|6$	13847.81	11488.02	Fixed	11947.85	7646.62	Fixed	NK	**12218.65**	5717.20	Fixed	11726.66	1183.14	**9934.33**	9293.01	11.8	15.3	23.9	26.2	4.0
	9	$LS\|10\|79\|5\|11$	15626.38	14373.26	Fixed	13747.58	9898.26	Fixed	NK	**14553.98**	7310.56	Fixed	12958.78	3600.00	11332.15	**11552.79**	6.9	17.1	20.6	12.2	11.0
	10	$LS\|6\|41\|5\|19$	11391.55	9564.52	Fixed	9705.43	**7764.34**	Fixed	NK	**10274.73**	4652.42	Fixed	9075.26	1338.74	7143.50	7184.45	9.8	20.3	24.4	16.9	11.7
																Average	**10.6**	**14.5**	**11.6**	**9.1**	**4.4**

UB, *LB*, *CT* stand for upper bound, lower bound, and computational time, respectively. NK stands for "Not Known".
Bold numbers represent *UB* and *LB* selected for a test problem.
Under average deviation column, LB_{BS} is determined in terms of *LB* related to RMILP model, MILP3 model, and B&P algorithm.
Under average deviation column, UB_{BS} is determined in terms of *UB* related to RMILP and MILP3 models.
B&P algorithm applies the DWD3 decomposition technique to decompose the problem.
UB_{GS} and LB_{GS} are determined based on the MILP3 model by relaxing desired bound constraints, i.e., $LB_{ih}^k = n_i$.

Table 25. The benefits of batching, performance of UB_{Alg} vs UB_{BS}, and performance of UB_{Alg} and UB_{BS} vs LB_{BS} for (Large, Large) level of problems

Level	Test Problem	$Level\|g\|\sum_{i\in g} n_i\|m\|\sum_{k\in m} v^k$	Group Scheduling (GS)			Batch Scheduling (BS)											Average Deviation				
			MILP Model			RMILP Model				MILP3 Model			Meta-heuristic		B&P						
			UB_{GS}	LB_{GS}	CT_{GS}	UB_{BS}	LB_{BS}	CT_{BS}	LB_{ih}^{k}	UB_{BS}	LB_{BS}	CT_{BS}	UB_{Alg}	CT_{Alg}	$B\&P_{DWD2}$	$B\&P_{DWD3}$	UB_{GS} vs UB_{BS}	UB_{GS} vs UB_{Alg}	LB_{BS} vs UB_{BS}	LB_{BS} vs UB_{Alg}	UB_{BS} vs UB_{Alg}
(Large, Large)	1	$LL\|9\|58\|5\|25$	19369.78	13544.87	Fixed	14539.04	4943.27	Fixed	NK	**15761.68**	7365.27	Fixed	14429.74	894.65	**11823.77**	10610.8	18.6	25.5	25.0	22.0	8.5
	2	$LL\|7\|51\|4\|12$	15600.57	12003.80	Fixed	10279.97	5037.19	Fixed	NK	**11012.16**	4139.91	Fixed	9995.10	2101.30	7955.93	**9158.261**	29.4	35.9	16.8	9.1	9.2
	3	$LL\|8\|58\|5\|8$	18279.95	12782.77	Fixed	10495.69	5247.85	Fixed	NK	**11991.34**	4508.02	Fixed	11080.98	1401.13	8741.88	**9260.098**	34.4	39.4	22.8	19.7	7.6
	4	$LL\|8\|62\|6\|7$	16371.37	11448.15	Fixed	10612.74	5943.13	Fixed	NK	**11235.26**	5350.12	Fixed	10669.76	833.87	7483.26	**7897.393**	31.4	34.8	29.7	35.1	5.0
	5	$LL\|8\|60\|2\|9$	22579.33	17373.59	Fixed	15020.31	4355.89	Fixed	NK	**15716.98**	7079.72	Fixed	13327.35	2021.88	11329.32	**12915.38**	30.4	41.0	17.8	3.2	15.2
	6	$LL\|10\|68\|3\|7$	17062.89	15352.50	Fixed	13747.26	4674.07	Fixed	NK	**14553.64**	5471.29	Fixed	13967.63	3596.90	11314.98	**11815.28**	14.7	18.1	18.8	18.2	4.0
	7	$LL\|7\|44\|6\|21$	20779.55	14947.28	Fixed	15594.41	5613.99	Fixed	NK	**16705.13**	7048.58	Fixed	14705.58	1818.48	**12222.20**	11411.37	19.6	29.2	26.8	20.3	12.0
	8	$LL\|8\|52\|4\|17$	14073.70	9982.51	Fixed	8654.66	4846.61	Fixed	NK	**9382.47**	3514.03	Fixed	8060.20	2098.06	6272.72	**6316.303**	33.3	42.7	32.7	27.6	14.1
	9	$LL\|8\|49\|7\|30$	20466.06	17388.77	Fixed	16102.24	6923.96	Fixed	NK	**17456.35**	6739.90	Fixed	14820.50	1049.43	11354.02	**12616.01**	14.7	27.6	27.7	17.5	15.1
	10	$LL\|8\|50\|6\|15$	15475.17	12527.82	Fixed	9796.40	2351.14	Fixed	NK	**10620.22**	3587.91	Fixed	9478.77	923.01	8352.02	**8727.133**	31.4	38.7	17.8	8.6	10.7
																Average	**25.8**	**33.3**	**23.6**	**18.1**	**10.1**

UB, *LB*, *CT* stand for upper bound, lower bound, and computational time, respectively. NK stands for “Not Known”.
Bold numbers represent *UB* and *LB* selected for a test problem.
Under average deviation column, LB_{BS} is determined in terms of *LB* related to RMILP model, MILP3 model, and B&P algorithm.
Under average deviation column, UB_{BS} is determined in terms of *UB* related to RMILP and MILP3 models.
B&P algorithm applies DWD3 approach to decompose the problem.
UB_{GS} and LB_{GS} are determined based on the MILP3 model by relaxing desired bound constraints, i.e., $LB_{ih}^{k} = n_i$.

- $LB_{selective} = \max\{Opt_{RD}, LB_{OL}, LB_{(B\&P_{DWD2})}, LB_{(B\&P_{DWD3})}\}$, when the optimal solution of the RMILP model violates LB_{ih}^{k}.
- $LB_{selective} = \max\{LB_{RD}, LB_{OL}, LB_{(B\&P_{DWD2})}, LB_{(B\&P_{DWD3})}\}$, when the optimal solutions of both the RMILP and MILP3 models are not attainable.

Thus, the results of the RMILP model, the MILP3 model, and the B&P algorithm (using DWD2 and DWD3 decomposition techniques) are considered to determine tight lower bound. In order to provide sufficient insight for arguments presented in Section 4.1.4.1, we reported the results of the RMILP model, the MILP3 model, and the B&P algorithm, where $UB_{Selective}$ and $LB_{Selective}$ are shown by bold numbers in Tables 23 through 23. For example, the RMILP model violates LB_{ih}^{k} for the sixth test problem in (small, small) level, but $LB_{Selective} = Opt_{RD}$ since $Opt_{RD} > LB_{OL} > LB_{(B\&P_{DWD3})} > LB_{(B\&P_{DWD2})}$. Contrary to this problem, since Opt_{RD} of the fifth test problem in (small, large) level violates LB_{ih}^{k} and $LB_{OL} < Opt_{RD} < LB_{(B\&P_{DWD2})} < LB_{(B\&P_{DWD3})}$, $LB_{Selective} = LB_{(B\&P_{DWD3})}$. Generally, one bold number in a row represents the optimal solution (i.e., $UB_{Selective}$=$LB_{Selective}$), while two bolds numbers in a row represent $UB_{Selective}$ and $LB_{Selective}$ of the test problem.

The following results 7 through 10 are obtained from Tables 23 through 25:

Result 7. the performance of the developed batch scheduling approach is compared with the group scheduling approach to uncover the benefits of batching. As a result of integrating the batching decision into group scheduling and, consequently, dropping the GTAs, Figure 26 show up to 34.4% and 42.7% reduction in the objective function value of group scheduling with the help of CPLEX and meta-heuristic algorithms, respectively. These benefits of batching become more pronounced as the size of the problems is increased, particularly in (large, large) level.

The result of a paired *t*-test in the following one-way hypothesis test applied to compare the average objective function value obtained by batch scheduling and group scheduling shows there is a statistically significant difference between the performances of batch scheduling and group scheduling ($P_{value} < 0.00001$).

$$\begin{cases} H_0: \mu_{BS} - \mu_{GS} = 0 \\ H_a: \mu_{BS} - \mu_{GS} < 0 \end{cases}$$

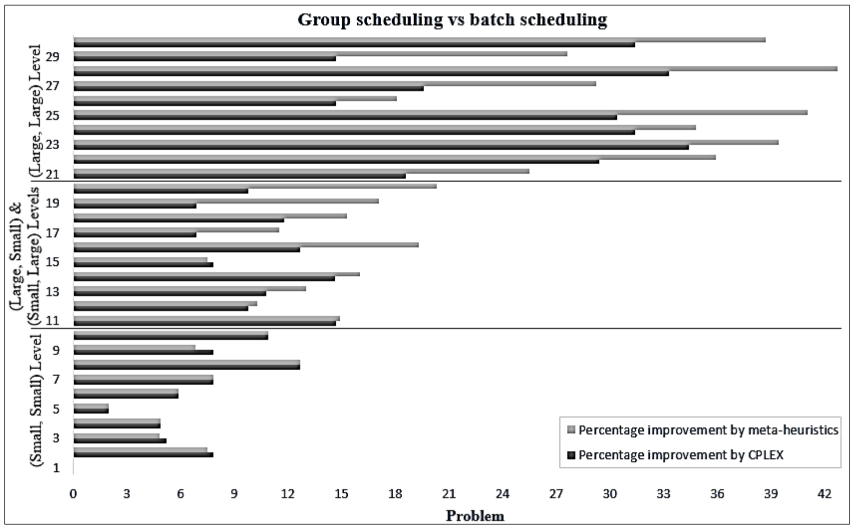


Figure 26. The benefits of integrating the batching decision into group scheduling

Result 8. the quality of the lower bound is of a vital importance. The quality of the lower bounds obtained from the B&P algorithm is more pronounced as the size of the problems is increased, particularly in (large, large) level. As seen in Tables 23 and 25, the lower bounds obtained from the B&P algorithm, either using DWD2 and DWD3, significantly outperform the lower bounds obtained from CPLEX, except for problems 3 and 10 in (small, large) and (large, small) levels, respectively. The average deviations between the lower bounds obtained from B&P and CPLEX are 17.93% and 77.92% for (small, large) & (large, small) and (large, large) levels, respectively. This notable improvement in the quality of the lower bounds increases up to 143.24% for problem 9 in (large, large) level.

Result 9. with respect to batch scheduling, the average deviation of meta-heuristic algorithms from the best lower bounds, i.e., $LB_{Selective}$, is 1.1%, 9.1%, and 18.1%, for (small, small), (small, large) & (large, small), and (large, large) level, respectively. Compared to 0.3%, 11.6%, and 23.6% average deviation between upper bounds of CPLEX and $LB_{Selective}$, meta-heuristic algorithms present a better performance, except for (small, small) level. With respect to (small, large) & (large, small) and (large, large) level, the deviation of CPLEX from meta-heuristic algorithms is 4.4% and 10.1% on average, which shows the superior performance of meta-heuristic algorithms, particularly for medium and large size problems.

Result 10. the RMILP model is capable of finding the optimal solutions and good quality lower bounds, particularly for small-size problems. Apart from this, the B&P algorithm using DWD2 is capable of finding better lower bounds compared to the B&P algorithm using DWD3 in some problems, i.e., problem 6 in (small, large) level, problem 8 in (large, small) level, and problems 1 and 7 in (large, large) level, since the number of violations on LB_{ih}^{k} is not significant.

8. CONCLUSIONS

Unlike the extensive amount of research on finding schedules for hybrid flow shop problems, little attention has been given to form inconsistent batches of jobs belonging to pre-determined groups to minimize any measure of performance. By disregarding group technology assumptions, there is the possibility of processing jobs belonging to a group in several batches, so that jobs belonging to the same group might be processed concurrently on more than one machine in a bottleneck stage. The hybrid flow shop batching and scheduling problem with the desired lower bounds on batch sizes, i.e., $HF_m\left|ST_{sd,f}, r_j, a_j, M_j, LB_i, skip\right|F_l(\alpha \sum w_j C_j, \beta \sum w_j T_j)$ is a complex problem with broad applications, particularly in many manufacturing industries. A bi-criteria objective function is considered to account for the benefits to both the producer and customers by minimizing a linear combination of total weighted completion time as well as total weighted tardiness of all jobs, subject to the mentioned operational constraints. The possibility of processing jobs belonging to a group in multiple batches is carefully investigated, when the number of jobs assigned to each batch does not violate the desired lower bounds on batch sizes. To create a balance between setup times and cumulative run times of developed batches on machines, the desired lower bounds on batch sizes are considered as the minimum number of jobs assigned to each batch. In order to depict the real industry requirements, further constraints of the problem are considered as dynamic job release times, dynamic machine availability times, different machine eligibilities and capabilities for job processing, and the possibility of stage skipping.

Four mathematical models have been developed to deal with the batch scheduling problem. The MILP1 and MILP2 models have been developed in two phases, i.e., batching and scheduling phases. The batching phase determines the optimal combination of batch compositions of all groups in all stages, with respect to the desired lower bounds on batch sizes, while the optimal assignment and sequence of batches on machines as well as sequence of jobs within batches are determined in the scheduling phase, regarding operational constraints and different combinations of batch compositions in the batching phase. The MILP1 model determines the job sequence and batch sequence by the precedence constraints between each pair of jobs within batches and each pair of developed batches on machines. The MILP2 model is very similar to the MILP1 model, except that it follows the position concept within batches to determine the job sequence within batches in the scheduling phase. Both MILP1 and MILP2 models have an extremely large solution space, particularly with regard to an exhaustive combination of enumerations between batch compositions. To combat this difficulty, the MILP3 and RMILP model are developed. The MILP3 model developed in terms of the flow conservation constraints integrates batching and scheduling phases of the MILP1 and MILP2 models. Therefore, the number of variables and, consequently, the complexity of the model is reduced. The RMILP model focuses on a non-dominated solution space by eliminating the batching phase.

However, the RMILP is also strongly NP-hard, and its effect is more pronounced on computational times as the size of the problem grows. Although the RMILP model cannot guarantee the optimal solution of other MILP models since it might violate the desired lower bounds on batch sizes, it is capable of developing good quality lower bounds in limited computational times, compared to other MILP models.

Since the optimal solution for large-size problems (that are commonly found in real industries) cannot be found within a reasonable computational time, several meta-heuristics have been developed as the main approach to deal with this problem to find the optimal/near optimal solutions, within an affordable time. A basic TS along with three TS meta-heuristics enhanced with a population-based structure, i.e., TS/PR, as well as a basic PSO along with three PSO meta-heuristics enhanced with a local search structure, i.e., PSO/LSA, are developed to deal with the proposed batch scheduling problem. TS/PR algorithms are different in terms of path construction techniques, while PSO/LSA are different in terms of the frequency of using LSA within the structure of PSO. An initial solution finding mechanism is developed to generate the initial populations. The refinement and adjustment steps are developed to change any infeasible solutions to feasible ones, during the search into the solution space. The optimal solution of batch scheduling in hybrid flow shop environments are not generally represented in the form of permutation sequences because there is a different combination between batch compositions of all groups in each stage of the HFS batch scheduling problem. Therefore, the permutation schedule does not hold true in the problem addressed in this research. Thus, both sets of hybrid meta-heuristics implement the stage-based interdependency strategy on local search and population-based structures to capture the move interdependency of jobs within stages and, consequently, increase the flexibility of batch scheduling.

In order to determine the performance of developed meta-heuristic algorithms, they should be compared through a robust measure. The best measure is the optimal solution or good quality lower bounds, which are attainable for small-size problems. In addition, with the help of lower bounding mechanism, tight lower bounds are obtained for medium- and large-size problems. A lower bounding mechanism based on column generation technique is developed, and embedded in the tree structure of a B&B algorithm, i.e., B&P algorithm, in order to develop tight lower bounds or optimally solve the HFS batch scheduling problem.

With the help of Dantzig-Wolfe Decomposition, the HFS problem is decomposed into a master problem and a set of sub-problems that are easier to solve. Three different decomposition techniques, DWD1, DWD2, and DWD3, are devised in this research in terms of the MILP1, RMILP, and MILP3 models, respectively, and the best among them, i.e., DWD3, has been used in column generation. Although each SP developed by DWD1 decomposition technique is smaller than the original problem, it has been shown that these sub-problems are still strongly NP-hard, and subsequently, cannot be solved to optimality within a reasonable amount of time. However, with the help of two virtual stages and eliminating the batching phase, the sub-problems are simplified. Then, by heuristically solving the sub-problems in early stages of the column generation algorithm, the sub-problems, and consequently, the restricted master problem have been reformulated in a way that they require drastically fewer variables and constraints. This being the case, the good quality lower bounds are obtained by this reformulation for small-size HFS problems. The SPs obtained by the DWD3 decomposition technique can be optimally solved, while DWD1 is only capable of finding lower bounds for small-size problems. Also, for larger size of problems, the SPs obtained by DWD1 are still NP-hard so that they cannot be optimally solved. Therefore, DWD1 is not capable of finding even good quality lower bound for large-size problems. In addition, the SPs obtained by DWD2 can be optimally solved but the desired lower bounds on batch sizes might be violated by some of the SPs. Therefore, DWD2 cannot guarantee to find tight lower bounds if the number of violations in batch sizes is significant. Thus, DWD2 and DWD3 are the only approaches that have been used in the B&P algorithm.

Compared to the group scheduling approach, the proposed batch scheduling approach achieves further improvement in the objective function value by taking the benefits of integrating the batching decision into the group scheduling approach, when the group technology assumptions (GTAs) are dropped. The superior quality of results obtained by the batch scheduling approach will encourage manufacturing industries to implement such a scheduling approach to determine the optimal/near optimal schedule. Based on a benchmark in the literature, the results of this comparison revealed that the batch scheduling approach is able to improve the results of the group scheduling approach up to 21% and even 1% improvement in manufacturing companies is significant. With the help of a comprehensive data generation mechanism, various experiments conducted on different set of problem structures confirm the superiority of PSO/LSA compared to TS/PR. A comparison between all developed algorithms revealed outstanding performance of TS/PR where PR explores trajectories connecting elite solutions based on a combination of LCS- & swap-based contractions (*Alg4*) as well as PSO/LSA where LSA is implemented for all particles within the population at each iteration (*Alg8*). The results indicate that a basic TS or PSO is not capable of finding good quality solution for the HFS batch scheduling problem. For medium- and large-size problems, where CPLEX is not able to find optimal solutions, the results of the best developed algorithm (*Alg4* and *Alg8*) are compared against the lower bounds obtained from the B&P algorithm and CPLEX. These comparisons revealed outstanding performance of developed meta-heuristic algorithms compared to CPLEX. Finally, the B&P algorithm is capable of finding tight lower bounds compared to the lower bounds obtained from CPLEX, particularly for large-size problems.

Some directions for future research are to extend the application of this research to other critical areas such as health care systems to improve the performance of their scheduling system. Also, due to the wide range of group scheduling applications, there will be growing interest on the improvement of the group scheduling approach by dropping the GTAs. i.e., batch scheduling. In realistic situations, as the workers learn how to perform a job, they will act faster in processing similar jobs. In the literature for the scheduling problems, this is referred to as the *learning effect* and represents the learning abilities of workers to perform similar jobs. Therefore, the runtime of jobs can be assumed to be dynamic. Finally, decomposing the problem into the SPs so that each SP is related to a machine in a stage, developing a master problem to create a connection between the SPs, and solving the problem optimally by the B&P algorithm in the basis of column generation can indeed be a challenging research issue worth pursuing in the future.

BIBLIOGRAPHY

Allahverdi, A. (2015). The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, 246(2), 345-378.

Allahverdi, A. and F. S. Al-Anzi. (2009). The two-stage assembly scheduling problem to minimize total completion time with setup times. *Computers & Operations Research*, 36(10), 2740-2747.

Allahverdi, A., C. Ng, T. E. Cheng and M. Y. Kovalyov. (2008). A survey of scheduling problems with setup times or costs. *European journal of operational research*, 187(3), 985-1032.

Armentano, V. c. A. and D. P. Ronconi. (1999). Tabu search for total tardiness minimization in flowshop scheduling problems. *Computers & operations research*, 26(3), 219-235.

Bai, J., Z.-R. Li and X. Huang. (2012). Single-machine group scheduling with general deterioration and learning effects. *Applied Mathematical Modelling*, 36(3), 1267-1274.

Barnhart, C., E. L. Johnson, G. L. Nemhauser, M. W. Savelsbergh and P. H. Vance. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations research*, 46(3), 316-329.

Basseur, M., F. Seynhaeve and E.-G. Talbi (2005). Path Relinking in Pareto Multi-objective Genetic Algorithms. EMO, Springer.

Bazaraa, M. S., J. J. Jarvis and H. D. Sherali (2011). Linear programming and network flows, John Wiley & Sons.

Behnamian, J., M. Zandieh and S. Fatemi Ghomi. (2010). A multi-phase covering Pareto-optimal front method to multi-objective parallel machine scheduling. *International Journal of Production Research*, 48(17), 4949-4976.

Behnamian, J., M. Zandieh and S. F. Ghomi. (2011). Bi-objective parallel machines scheduling with sequence-dependent setup times using hybrid metaheuristics and weighted min–max technique. *Soft Computing*, 15(7), 1313-1331.

Ben-Daya, M. and M. Al-Fawzan. (1998). A tabu search approach for the flow shop scheduling problem. *European journal of operational research*, 109(1), 88-95.

Bozorgirad, M. A. (2013). Bi-criteria group scheduling with learning in hybrid flow shops (Doctoral Dissertation), Oregon State University, Oregon, U.S.A.

Bozorgirad, M. A. and R. Logendran. (2012). Sequence-dependent group scheduling problem on unrelated-parallel machines. *Expert Systems with Applications*, 39(10), 9021-9030.

Bozorgirad, M. A. and R. Logendran. (2013). Bi-criteria group scheduling in hybrid flowshops. *International Journal of Production Economics*, 145(2), 599-612.

Bozorgirad, M. A. and R. Logendran (2014). Developing tight lower bounds for hybrid flow shop scheduling problems. IIE annual conference proceedings. Montreal: Institute of industrial engineers.

Brah, S. A. and J. L. Hunsucker. (1991). Branch and bound algorithm for the flow shop with multiple processors. *European journal of operational research*, 51(1), 88-99.

Bülbül, K., P. Kaminsky and C. Yano. (2004). Flow shop scheduling with earliness, tardiness, and intermediate inventory holding costs. *Naval Research Logistics (NRL)*, 51(3), 407-445.

Chang, W. (2000). 7.2: Invited Paper: Fourth-Generation TFT-LCD Production Line. SID Symposium Digest of Technical Papers, Wiley Online Library.

Chen, C.-L. and C.-L. Chen. (2009). Bottleneck-based heuristics to minimize total tardiness for the flexible flow line with unrelated parallel machines. *Computers & Industrial Engineering*, 56(4), 1393-1401.

Chen, Z.-L. and C.-Y. Lee. (2002). Parallel machine scheduling with a common due window. *European Journal of Operational Research*, 136(3), 512-527.

Chen, Z.-L. and W. B. Powell. (1999a). A column generation based decomposition algorithm for a parallel machine just-in-time scheduling problem. *European Journal of Operational Research*, 116(1), 220-232.

Chen, Z.-L. and W. B. Powell. (1999b). Solving parallel machine scheduling problems by column generation. *INFORMS Journal on Computing*, 11(1), 78-94.

Chen, Z. L. and W. B. Powell. (2003). Exact algorithms for scheduling multiple families of jobs on parallel machines. *Naval Research Logistics (NRL)*, 50(7), 823-840.

Cheng, R., M. Gen and Y. Tsujimura. (1999). A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies. *Computers & Industrial Engineering*, 36(2), 343-364.

Choi, H.-S., J.-S. Kim and D.-H. Lee. (2011). Real-time scheduling for reentrant hybrid flow shops: A decision tree based mechanism and its application to a TFT-LCD line. *Expert systems with Applications*, 38(4), 3514-3521.

Choobineh, F. F., E. Mohebbi and H. Khoo. (2006). A multi-objective tabu search for a single-machine scheduling problem with sequence-dependent setup times. *European Journal of Operational Research*, 175(1), 318-337.

Cormen, T., C. Leiserson, R. Rivest and C. Stein. (1990). Introduction to Algorithms, The MIT Electrical Engineering and Computer Science Series.

CPLEX, I. I. (2009). V12. 1: User's Manual for CPLEX. *International Business Machines Corporation*, 46(53), 157.

Dantzig, G. B. and P. Wolfe. (1960). Decomposition principle for linear programs. *Operations research*, 8(1), 101-111.

Desrosiers, J., F. Soumis and M. Desrochers. (1984). Routing with time windows by column generation. *Networks*, 14(4), 545-565.

Dessouky, M. M., M. I. Dessouky and S. K. Verma. (1998). Flowshop scheduling with identical jobs and uniform parallel machines. *European Journal of Operational Research*, 109(3), 620-631.

Du, J. and J. Y.-T. Leung. (1990). Minimizing total tardiness on one machine is NP-hard. *Mathematics of operations research*, 15(3), 483-495.

Du Merle, O., D. Villeneuve, J. Desrosiers and P. Hansen. (1997). Stabilisation dans le cadre de la génération de colonnes. *Cahiers du GERAD*,

Dugardin, F., F. Yalaoui and L. Amodeo. (2010). New multi-objective method to solve reentrant hybrid flow shop scheduling problem. *European Journal of Operational Research*, 203(1), 22-31.

Eberhart, R. and J. Kennedy. (1995). Particle swarm optimization, proceeding of IEEE International Conference on Neural Network. *Perth, Australia*, 1942-1948.

Eren, T. (2007). A multicriteria flowshop scheduling problem with setup times. *Journal of Materials Processing Technology*, 186(1), 60-65.

Eren, T. and E. Güner. (2006). A bicriteria scheduling with sequence-dependent setup times. *Applied Mathematics and Computation*, 179(1), 378-385.

Farley, A. A. (1990). A note on bounding a class of linear programming problems, including cutting stock problems. *Operations Research*, 38(5), 922-923.

Garey, M. and D. Johnson (1979). Computers and intractability. New York: H, Freeman and Company.

Garey, M. R., D. S. Johnson and R. Sethi. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, 1(2), 117-129.

Gelogullari, C. A. and R. Logendran. (2010). Group-scheduling problems in electronics manufacturing. *Journal of Scheduling*, 13(2), 177-202.

Gilmore, P. C. and R. E. Gomory. (1961). A linear programming approach to the cutting-stock problem. *Operations research*, 9(6), 849-859.

Gilmore, P. C. and R. E. Gomory. (1963). A linear programming approach to the cutting stock problem-Part II. *Operations research*, 11(6), 863-888.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5), 533-549.

Glover, F. (1997). Tabu search and adaptive memory programming—advances, applications and challenges. Interfaces in computer science and operations research, Springer: 1-75.

Glover, F., M. Laguna and R. Martí. (2000). Fundamentals of scatter search and path relinking. *Control and cybernetics*, 29(3), 653-684.

Graham, R. L., E. L. Lawler, J. K. Lenstra and A. R. Kan. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5, 287-326.

Guinet, A. (1993). Scheduling sequence-dependent jobs on identical parallel machines to minimize completion time criteria. *The International Journal of Production Research*, 31(7), 1579-1594.

Gupta, J., A. Hariri and C. Potts. (1997). Scheduling a two-stage hybrid flow shop with parallel machines at the first stage. *Annals of Operations Research*, 69, 171-191.

Gupta, J. N. and S. Chantaravarapan. (2008). Single machine group scheduling with family setups to minimize total tardiness. *International Journal of Production Research*, 46(6), 1707-1722.

Gupta, J. N. and W. P. Darrow. (1986). The two-machine sequence dependent flowshop scheduling problem. *European Journal of Operational Research*, 24(3), 439-446.

Hajinejad, D., N. Salmasi and R. Mokhtari. (2011). A fast hybrid particle swarm optimization algorithm for flow shop sequence dependent group scheduling problem. *Scientia Iranica*, 18(3), 759-764.

Haouari, M., L. Hidri and A. Gharbi. (2006). Optimal scheduling of a two-stage hybrid flow shop. *Mathematical Methods of Operations Research*, 64(1), 107-124.

Hendizadeh, S. H., H. Faramarzi, S. A. Mansouri, J. N. Gupta and T. Y. ElMekkawy. (2008). Meta-heuristics for scheduling a flowline manufacturing cell with sequence dependent family setup times. *International journal of production economics*, 111(2), 593-605.

Ho, J. C. and Y.-L. Chang. (1995). Minimizing the number of tardy jobs for m parallel machines. *European Journal of Operational Research*, 84(2), 343-355.

Ho, Y.-C. and T.-S. Su (2010). The machine layout within a TFT-LCD bay with an in-line stocker system and an RGV system. Computers and Industrial Engineering (CIE), 2010 40th International Conference on, IEEE.

Hooker, J. N. and G. Ottosson. (2003). Logic-based Benders decomposition. *Mathematical Programming*, 96(1), 33-60.

Hooker, J. N. and H. Yan. (1995). Logic circuit verification by Benders decomposition. *Principles and practice of constraint programming: The Newport Papers*, 267-288.

Hsu, H.-M., Y. Hsiung, Y.-Z. Chen and M.-C. Wu. (2009). A GA methodology for the scheduling of yarn-dyed textile production. *Expert Systems with Applications*, 36(10), 12095-12103.

Jang, Y. J. and G.-H. Choi (2006). Introduction to automated material handling systems in LCD panel production lines. Automation Science and Engineering, 2006. CASE'06. IEEE International Conference on, IEEE.

Jeong, B., S.-W. Kim and Y.-J. Lee. (2001). An assembly scheduler for TFT LCD manufacturing. *Computers & Industrial Engineering*, 41(1), 37-58.

Jia, S. and Z.-H. Hu. (2014). Path-relinking Tabu search for the multi-objective flexible job shop scheduling problem. *Computers & Operations Research*, 47, 11-26.

Jin, Z., K. Ohno, T. Ito and S. Elmaghraby. (2002). Scheduling hybrid flowshops in printed circuit board assembly lines. *Production and Operations Management*, 11(2), 216-230.

Jungwattanakit, J., M. Reodecha, P. Chaovalitwongse and F. Werner. (2009). A comparison of scheduling algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria. *Computers & Operations Research*, 36(2), 358-378.

Kang, Y.-H., S.-S. Kim and H. J. Shin. (2007). A scheduling algorithm for the reentrant shop: an application in semiconductor manufacture. *The International Journal of Advanced Manufacturing Technology*, 35(5), 566-574.

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

Keshavarz, T. and N. Salmasi. (2013). Makespan minimisation in flexible flowshop sequence-dependent group scheduling problem. *International Journal of Production Research*, 51(20), 6182-6193.

Keshavarz, T. and N. Salmasi. (2014). Efficient upper and lower bounding methods for flowshop sequence-dependent group scheduling problems. *European Journal of Industrial Engineering*, 8(3), 366-387.

Kim, D.-W., K.-H. Kim, W. Jang and F. F. Chen. (2002). Unrelated parallel machine scheduling with setup times using simulated annealing. *Robotics and Computer-Integrated Manufacturing*, 18(3), 223-231.

Kim, S.-I., H.-S. Choi and D.-H. Lee (2006). Tabu search heuristics for parallel machine scheduling with sequence-dependent setup and ready times. International Conference on Computational Science and Its Applications, Springer.

Kim, S., K.-N. Chang and J.-Y. Lee. (1995). A descent method with linear programming subproblems for nondifferentiable convex optimization. *Mathematical programming*, 71(1), 17-28.

Laguna, M., J. W. Barnes and F. Glover. (1993). Intelligent scheduling with tabu search: an application to jobs with linear delay penalties and sequence-dependent setup costs and times. *Applied Intelligence*, 3(2), 159-172.

Laguna, M., J. W. Barnes and F. W. Glover. (1991). Tabu search methods for a single machine scheduling problem. *Journal of Intelligent Manufacturing*, 2(2), 63-73.

Lasdon, L. S. (1970). Optimization theory for large systems, Courier Corporation.

Lee, J.-H., J.-M. Yu and D.-H. Lee. (2013). A tabu search algorithm for unrelated parallel machine scheduling with sequence-and machine-dependent setups: minimizing total tardiness. *The International Journal of Advanced Manufacturing Technology*, 69(9-12), 2081-2089.

Lenstra, J. K., D. B. Shmoys and É. Tardos. (1990). Approximation algorithms for scheduling unrelated parallel machines. *Mathematical programming*, 46(1), 259-271.

Li, S., C. Ng and J. Yuan. (2011). Group scheduling and due date assignment on a single machine. *International Journal of Production Economics*, 130(2), 230-235.

Li, X., M. Baki and Y. P. Aneja. (2010). An ant colony optimization metaheuristic for machine–part cell formation problems. *Computers & Operations Research*, 37(12), 2071-2081.

Lian, Z., X. Gu and B. Jiao. (2006). A similar particle swarm optimization algorithm for permutation flowshop scheduling to minimize makespan. *Applied mathematics and computation*, 175(1), 773-785.

Lian, Z., X. Gu and B. Jiao. (2008). A novel particle swarm optimization algorithm for permutation flow-shop scheduling to minimize makespan. *Chaos, Solitons & Fractals*, 35(5), 851-861.

Liou, C.-D., Y.-C. Hsieh and Y.-Y. Chen. (2013). A new encoding scheme-based hybrid algorithm for minimising two-machine flow-shop group scheduling problem. *International Journal of Systems Science*, 44(1), 77-93.

Liou, C.-D. and C.-H. Liu. (2010). A novel encoding scheme of PSO for two-machine group scheduling. *Advances in Swarm Intelligence*, 128-134.

Liu, B., L. Wang and Y.-H. Jin. (2008). An effective hybrid PSO-based algorithm for flow shop scheduling with limited buffers. *Computers & Operations Research*, 35(9), 2791-2806.

Liu, J., Y. Wang and X. Min. (2014). Single-machine scheduling with common due-window assignment for deteriorating jobs. *Journal of the Operational Research Society*, 65(2), 291-301.

Liu, Y. and I. Karimi. (2008). Scheduling multistage batch plants with parallel units and no interstage storage. *Computers & Chemical Engineering*, 32(4), 671-693.

Logendran, R., S. Carson and E. Hanson. (2005). Group scheduling in flexible flow shops. *International Journal of Production Economics*, 96(2), 143-155.

Logendran, R., P. deSzoeke and F. Barnard. (2006a). Sequence-dependent group scheduling problems in flexible flow shops. *International Journal of Production Economics*, 102(1), 66-86.

Logendran, R., B. McDonell and B. Smucker. (2007). Scheduling unrelated parallel machines with sequence-dependent setups. *Computers & Operations Research*, 34(11), 3420-3438.

Logendran, R., N. Salmasi and C. Sriskandarajah. (2006b). Two-machine group scheduling problems in discrete parts manufacturing with sequence-dependent setups. *Computers & Operations Research*, 33(1), 158-180.

Logendran, R. and A. Sonthinen. (1997). A tabu search-based approach for scheduling job-shop type flexible manufacturing systems. *Journal of the Operational Research Society*, 48(3), 264-277.

Logendran, R. and F. Subur. (2004). Unrelated parallel machine scheduling with job splitting. *IIE Transactions*, 36(4), 359-372.

Lopes, M. J. P. and J. V. de Carvalho. (2007). A branch-and-price algorithm for scheduling parallel machines with sequence dependent setup times. *European journal of operational research*, 176(3), 1508-1527.

Low, C. and W.-Y. Lin. (2012). Single machine group scheduling with learning effects and past-sequence-dependent setup times. *International Journal of Systems Science*, 43(1), 1-8.

Lu, D. and R. Logendran. (2013). Bi-criteria group scheduling with sequence-dependent setup time in a flow shop. *Journal of the Operational Research Society*, 64(4), 530-546.

Lübbecke, M. E. and J. Desrosiers. (2005). Selected topics in column generation. *Operations Research*, 53(6), 1007-1023.

Luo, H., G. Q. Huang, Y. Shi, T. Qu and Y. F. Zhang. (2012). Hybrid flowshop scheduling with family setup time and inconsistent family formation. *International Journal of Production Research*, 50(6), 1457-1475.

Luo, J. and Y. Hu (2013). A new GRASP and path relinking for single machine scheduling with sequence dependent setups. Control and Automation (ICCA), 2013 10th IEEE International Conference on, IEEE.

Manne, A. S. (1960). On the job-shop scheduling problem. *Operations Research*, 8(2), 219-223.

Marsten, R. E., W. Hogan and J. W. Blankenship. (1975). The boxstep method for large-scale optimization. *Operations Research*, 23(3), 389-405.

Mehravaran, Y. (2013). Hybrid Flowshop Scheduling with Dual Resources in a Supply Chain (Doctoral Dissertation), Oregon State University, Oregon, U.S.A.

Mehravaran, Y. and R. Logendran. (2011). Bicriteria supply chain scheduling on unrelated-parallel machines. *Journal of the Chinese Institute of Industrial Engineers*, 28(2), 91-101.

Mehravaran, Y. and R. Logendran. (2012). Non-permutation flowshop scheduling in a supply chain with sequence-dependent setup times. *International Journal of Production Economics*, 135(2), 953-963.

Montgomery, D. C. (2009). Introduction to statistical quality control, John Wiley & Sons (New York).

Neammanee, P. and M. Reodecha. (2009). A memetic algorithm-based heuristic for a scheduling problem in printed circuit board assembly. *Computers & Industrial Engineering*, 56(1), 294-305.

Nowicki, E. and C. Smutnicki. (1996). A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research*, 91(1), 160-175.

Pacheco, J., F. Ángel-Bello and A. Álvarez. (2013). A multi-start tabu search method for a single-machine scheduling problem with periodic maintenance and sequence-dependent set-up times. *Journal of Scheduling*, 16(6), 661-673.

Pan, Q.-K., M. F. Tasgetiren and Y.-C. Liang. (2008). A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem. *Computers & Operations Research*, 35(9), 2807-2839.

Pan, Q. and L. Wang. (2008). A novel multi-objective particle swarm optimization algorithm for no-wait flow shop scheduling problems. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 222(4), 519-539.

Pandya, V. and R. Logendran (2010). Weighted tardiness minimization in flexible flowshops. IIE Annual Conference. Proceedings, Institute of Industrial and Systems Engineers (IISE).

Peng, B., Z. Lü and T. Cheng. (2015). A tabu search/path relinking algorithm to solve the job shop scheduling problem. *Computers & Operations Research*, 53, 154-164.

Poli, R., J. Kennedy and T. Blackwell. (2007). Particle swarm optimization. *Swarm intelligence*, 1(1), 33-57.

Potts, C. and L. Van Wassenhove. (1992). Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity. *Journal of the Operational Research Society*, 395-406.

Potts, C. N. and M. Y. Kovalyov. (2000). Scheduling with batching: A review. *European journal of operational research*, 120(2), 228-249.

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(2), 137-143.

Rana, S. and N. Singh. (1994). Group scheduling jobs on a single machine: A multi-objective approach with preemptive priority structure. *European Journal of Operational Research*, 79(1), 38-50.

Ribas-Vila, I., R. Companys-Pascual and M. Mateo-Doll. (2009). Bicriteria scheduling problem on parallel machine. *DYNA*, 84(5), 429-440.

Rousseau, L.-M., M. Gendreau and D. Feillet. (2007). Interior point stabilization for column generation. *Operations Research Letters*, 35(5), 660-668.

Ruiz, R. and C. Maroto. (2006). A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. *European Journal of Operational Research*, 169(3), 781-800.

Ruiz, R., F. S. Şerifoğlu and T. Urlings. (2008). Modeling realistic hybrid flexible flowshop scheduling problems. *Computers & Operations Research*, 35(4), 1151-1175.

Ruiz, R. and J. A. Vázquez-Rodríguez. (2010). The hybrid flow shop scheduling problem. *European Journal of Operational Research*, 205(1), 1-18.

Salmasi, N., R. Logendran and M. R. Skandari. (2010). Total flow time minimization in a flowshop sequence-dependent group scheduling problem. *Computers & Operations Research*, 37(1), 199-212.

Salmasi, N., R. Logendran and M. R. Skandari. (2011). Makespan minimization of a flowshop sequence-dependent group scheduling problem. *The International Journal of Advanced Manufacturing Technology*, 56(5), 699-710.

Sawik, T. (2000). Mixed integer programming for scheduling flexible flow lines with limited intermediate buffers. *Mathematical and Computer Modelling*, 31(13), 39-52.

Schaller, J. E., J. N. Gupta and A. J. Vakharia. (2000). Scheduling a flowline manufacturing cell with sequence dependent family setup times. *European Journal of Operational Research*, 125(2), 324-339.

Schiavinotto, T. and T. Stützle. (2007). A review of metrics on permutations for search landscape analysis. *Computers & operations research*, 34(10), 3143-3153.

Shahvari, O. and R. Logendran. (2015). Bi-Criteria Batch Scheduling on Unrelated-Parallel Machines. *Proceedings of the 2015 Industrial and Systems Engineering Research Conference (ISERC) (CD-ROM)*, Nashville, Tennessee, USA.

Shahvari, O. and R. Logendran. (2016a). Bi-Criteria Batch Scheduling in Hybrid Flow Shop. *Proceedings of the 2016 Industrial and Systems Engineering Research Conference (ISERC) (CD-ROM)*, Anaheim, California, USA.

Shahvari, O. and R. Logendran. (2016b). Hybrid flow shop batching and scheduling with a bi-criteria objective. *International Journal of Production Economics*, 179, 239-258.

Shahvari, O. and R. Logendran. (2017). An Enhanced tabu search algorithm to minimize a bi-criteria objective in batching and scheduling problems on unrelated-parallel machines with desired lower bounds on batch sizes. *Computers & Operations Research*, 77, 154-176.

Shahvari, O., N. Salmasi, R. Logendran and B. Abbasi. (2012). An efficient tabu search algorithm for flexible flow shop sequence-dependent group scheduling problems. *International Journal of Production Research*, 50(15), 4237-4254.

Shen, L., J. N. Gupta and U. Buscher. (2014). Flow shop batching and scheduling with sequence-dependent setup times. *Journal of scheduling*, 17(4), 353-370.

Shin, H. and V. Leon. (2004). Scheduling with product family set-up times: an application in TFT LCD manufacturing. *International journal of production research*, 42(20), 4235-4248.

Sun, X., J. S. Noble and C. M. Klein. (1999). Single-machine scheduling with sequence dependent setup to minimize total weighted squared tardiness. *IIE transactions*, 31(2), 113-124.

Suresh, V. and D. Chaudhuri. (1994). Minimizing maximum tardiness for unrelated parallel machines. *International Journal of Production Economics*, 34(2), 223-229.

Tadayon, B. and N. Salmasi. (2013). A two-criteria objective function flexible flowshop scheduling problem with machine eligibility constraint. *The International Journal of Advanced Manufacturing Technology*, 1-15.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *european journal of operational research*, 64(2), 278-285.

Tang, L.-x. and H. Xuan. (2006). Lagrangian relaxation algorithms for real-time hybrid flowshop scheduling with finite intermediate buffers. *Journal of the Operational Research Society*, 57(3), 316-324.

Tasgetiren, M. F., Y.-C. Liang, M. Sevkli and G. Gencyilmaz (2004a). Particle swarm optimization algorithm for makespan and maximum lateness minimization in permutation flowshop sequencing problem. Proceedings of the fourth international symposium on intelligent manufacturing systems, Sakarya, Turkey.

Tasgetiren, M. F., Y.-C. Liang, M. Sevkli and G. Gencyilmaz. (2007). A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *European journal of operational research*, 177(3), 1930-1947.

Tasgetiren, M. F., M. Sevkli, Y.-C. Liang and G. Gencyilmaz (2004b). Particle swarm optimization algorithm for single machine total weighted tardiness problem. Evolutionary Computation, 2004. CEC2004. Congress on, IEEE.

Tavakkoli-Moghaddam, R., N. Javadian, A. Khorrami and Y. Gholipour-Kanani. (2010). Design of a scatter search method for a novel multi-criteria group scheduling problem in a cellular manufacturing system. *Expert Systems with Applications*, 37(3), 2661-2669.

Tseng, C.-T. and C.-J. Liao. (2008). A particle swarm optimization algorithm for hybrid flow-shop scheduling with multiprocessor tasks. *International Journal of Production Research*, 46(17), 4655-4670.

Uzsoy, R., C.-Y. Lee and L. A. Martin-Vega. (1992). A review of production planning and scheduling models in the semiconductor industry part I: system characteristics, performance evaluation and production planning. *IIE transactions*, 24(4), 47-60.

Vallada, E., R. Ruiz and J. M. Framinan. (2015). New hard benchmark for flowshop scheduling problems minimising makespan. *European Journal of Operational Research*, 240(3), 666-677.

Van Den Akker, J., C. A. Hurkens and M. W. Savelsbergh. (2000). Time-indexed formulations for machine scheduling problems: Column generation. *INFORMS Journal on Computing*, 12(2), 111-124.

Van Den Akker, J. M., J. A. Hoogeveen and S. L. van de Velde. (1999). Parallel machine scheduling by column generation. *Operations Research*, 47(6), 862-872.

Vanderbeck, F. (1994). Decomposition and column generation for integer programs, Université catholique de Louvain.

Vanderbeck, F. (2000). On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. *Operations Research*, 48(1), 111-128.

Vanderbeck, F. and L. A. Wolsey. (1996). An exact algorithm for IP column generation. *Operations research letters*, 19(4), 151-159.

Varmazyar, M. and N. Salmasi. (2012). Sequence-dependent flow shop scheduling problem minimising the number of tardy jobs. *International Journal of Production Research*, 50(20), 5843-5858.

Wang, D., M. Gen and R. Cheng. (1999). Scheduling grouped jobs on single machine with genetic algorithm. *Computers & industrial engineering*, 36(2), 309-324.

Wang, D., Y. Huo and P. Ji. (2014). Single-machine group scheduling with deteriorating jobs and allotted resource. *Optimization Letters*, 8(2), 591-605.

Wang, J.-B. and J.-J. Wang. (2014). Single machine group scheduling with time dependent processing times and ready times. *Information Sciences*, 275, 226-231.

Webster, S. and K. R. Baker. (1995). Scheduling groups of jobs on a single machine. *Operations Research*, 43(4), 692-703.

Wein, L. M. (1988). Scheduling semiconductor wafer fabrication. *IEEE Transactions on semiconductor manufacturing*, 1(3), 115-130.

Wilhelm, W. E. (2001). A technical review of column generation in integer programming. *Optimization and Engineering*, 2(2), 159-200.

Xu, D. and Y. Yin. (2011). Comments on "A bicriteria flowshop scheduling problem with setup times". *Applied Mathematics and Computation*, 217(17), 7361-7364.

Yaurima, V., L. Burtseva and A. Tchernykh. (2009). Hybrid flowshop with unrelated machines, sequence-dependent setup time, availability constraints and limited buffers. *Computers & Industrial Engineering*, 56(4), 1452-1463.

Yazdani Sabouni, M. and R. Logendran. (2013). A single machine carryover sequence-dependent group scheduling in PCB manufacturing. *Computers & Operations Research*, 40(1), 236-247.

Zandieh, M., S. F. Ghomi and S. M. Husseini. (2006). An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times. *Applied Mathematics and Computation*, 180(1), 111-127.

Zandieh, M. and N. Karimi. (2011). An adaptive multi-population genetic algorithm to solve the multi-objective group scheduling problem in hybrid flexible flowshop with sequence-dependent setup times. *Journal of Intelligent Manufacturing*, 22(6), 979-989.

Zeng, R.-Q., M. Basseur and J.-K. Hao. (2013). Solving bi-objective flow shop problem with hybrid path relinking algorithm. *Applied Soft Computing*, 13(10), 4118-4132.

APPENDICES

Appendix A. The solution space of the RMILP model vs. the original MILP model

By recognizing that ${}_{n_i}C_r$ represents the total number of batch compositions of group i with n_i^k jobs in stage k, when n_i^k jobs should be assigned to r batches ($r \in S_i, S_i = \{1, \dots, \max_{h \in V^k} \lceil n_i^k / LB_{ih}^k \rceil\}$), all possible combinations ${}_{n_i}C_r, \forall i \in G, r \in S_i$, are determined as follows:

$$ {}_{n_i^k}C_1 = \binom{n_i^k}{n_i^k} \qquad \rightarrow \text{if there is only one batch.} $$

$$ {}_{n_i^k}C_2 = \binom{n_i^k}{1} + \binom{n_i^k}{2} + \cdots + \binom{n_i^k}{\lfloor n_i^k/2 \rfloor} / \left(2^{\left\lfloor \lfloor n_i^k/2 \rfloor / n_i^k/2 \right\rfloor}\right) \qquad \rightarrow \text{if there are only two batches.} $$

$$ \vdots \qquad\qquad\qquad \vdots $$

$$ {}_{n_i^k}C_{n_i^k} = \left(\binom{n_i^k}{1} \binom{n_i^k - 1}{1} \cdots \binom{2}{1} \right) / n_i^k! \qquad \rightarrow \text{if there are } n_i^k \text{ batches.} $$

Ceiling brackets $\lfloor . \rfloor$ rounds number to a lower integer. Then, the total number of batch compositions of group i in stage k, i.e., BC_i, is determined as follows:

$$ BC_i^k = \sum_{r \in S_i} {}_{n_i^k}C_r $$

As a result, there is an enormous number of combinations (BC^k) between BC_i^k of all groups in stage k, which is determined as follows:

$$ BC^k = \prod_{i \in I^k} BC_i^k = \prod_{i \in I^k} \left(\sum_{r \in S_i} {}_{n_i^k}C_r \right) $$

Consequently, an exhaustive combination enumeration between BC_i^k of all groups in all stages, i.e., BC_{Total}, is determined as follows:

$$ BC^{Total} = \prod_{i \in I^k} BC^k = \prod_{k \in M} \left(\prod_{i \in I^k} \left(\sum_{r \in S_i} {}_{n_i^k}C_r \right) \right) $$

For example in a two-stage problem including 3 groups with 3 jobs each (without job skipping and $LB_{ih}^k =$ 1), $n_i^k = 3$ and $BC_i^k = \sum_{r=1}^{3} {}_3C_r = 5$ ($\forall\, i = 1, 2, 3; \forall\, k = 1, 2$) and, subsequently, $BC^{Total} =$

$\prod_{k=1}^{2}\prod_{i=1}^{3}(5) = 15{,}625$. By increasing g and n_i^k by one each ($g = 4$ and $n_i^k = 4$), $BC^{Total} = \prod_{k=1}^{2}\prod_{i=1}^{4}(9) = 43{,}046{,}721$.

Since in the original MILP models, particularly the MILP1 and MILP2 models, the best sequence should be determined for each member of BC^{Total} to determine the optimal solution, the solution space increases exponentially, which leads to requiring unaffordable solution space and unreasonable computational time. The identification of structural non-dominance properties (usually corresponding to the batch composition restrictions in this research) is a key step to reduce the solution space to a non-dominated set and, consequently, to make possible the solution in affordable time.

Proposition: *By restricting the batching phase of batch scheduling to allocate one and only one job to each batch of each group, the optimal solution is guaranteed when there is no violation on desired lower bounds on batch sizes.*

Proof. Batch scheduling addressed here has multiple optimal solutions so that all optimal solutions have the same job assignments and job sequences on machines. The only difference is related to different combination between batch compositions in the entire stages, leading to different batch assignments and sequences on machines as well as different job sequences within batches. Apart from this, as long as job assignments and job sequences on machines of a schedule are not changed, the completion time of jobs and, subsequently, their tardiness are not changed, irrespective of which batch a job is assigned. Also, there is no setup time between batches belonging to the same group.

With the help of these arguments, it is concluded that these multiple optimal solutions can be converted to each other by changing at least one batch composition of a group related to a stage. As a result, if $LB_{ih}^{k} = 1$ ($\forall i \in I^k; h \in V^k; k \in M$), then each optimal schedule (S^{Opt}) can be converted to the specific optimal schedule ($S_{Relaxed}^{Opt}$), by decomposing and replacing each batch including more than one job in S^{Opt} to several batches including only one job in $S_{Relaxed}^{Opt}$, for the entire stages. $S_{Relaxed}^{Opt}$ restricts the batching phase to allocate one and only one job to each batch of each group in each stage. Therefore, based on $S_{Relaxed}^{Opt}$, the RMILP model is developed so that it guarantees to obtain the optimal solution in fraction of the computation time required by the original MILP model, because of focusing on the non-dominated solution space. This being the case, the batching phase is eliminated from batch scheduling and, consequently the optimal sequence of the RMILP model ($S_{Relaxed}^{Opt}$) can be interpreted as the optimal sequences for the original MILP model with the help of merging sequential batches of the same group as a single batch(es) on each machine. An extension of the developed proposition is that the RMILP model presents the optimal solution of the original MILP model as long as the desired lower bounds are not violated by $S_{Relaxed}^{Opt}$.

Therefore, non-dominated solution space as a structural non-dominance property is identified based on the batch composition restrictions to reduce the solution space and, consequently, to make possible either the optimal solutions or good quality lower bounds for problems in affordable computational times. This being the case, the RMILP model is developed by relaxing the desired lower bounds on batch sizes (i.e., $LB_{ih}^{k} = 1; \forall i \in I^{k}; h \in V^{k}; k \in K$) and eliminating the job assignment to batches, i.e., the batching phase.

Appendix B. The RMILP model based on the MILP2 model

Sets and Indices

G	Set of groups, indexed by i, p	$G = \{1,2,\dots,g\}$
G_i	Set of jobs of group i, indexed by j, q	$G_i = \{1,2,\dots,n_i\}$
K	Set of stages, indexed by k	$K = \{1,2,\dots,m\}$
V^k	Set of machines in stage k, indexed by h	$V^k = \{1,2,\dots,v^k\}$

Subsets

I^k	Subset of groups, which must be processed in stage k	$I^k \subset G$
J_i^k	Subset of jobs of group i, which must be processed in stage k	$J_i^k \subset G_i$
V_{ij}^k	Subset of machines in stage k, which can process job j of group i	$V_{ij}^k \subset V^k$
K_{ij}	Subset of stages in an ascending order, which must be visited by job j of group i	$K_{ij} \subset K$

Parameters

g	Number of groups
n_i	Number of jobs of group i
n_i^k	Number of jobs of group i, which must be processed in stage k
m	Number of stages
v^k	Number of machines in stage k
m_{ij}	Number of stages, which must be visited by job j of group i
$st_{ij(l)}$	l^{th} stage among subset K_{ij}
t_{ijh}^k	Run time of job j of group i on machine h in stage k
S_{pih}^k	Required setup time to process any batch of group i on machine h in stage k if a batch of group p is the preceding batch ($p = 0$ refers to the reference batch)
d_{ij}	Due date of job j of group i
r_{ij}	Release time of job j of group i
w_{ij}	Weight of job j of group i
a_h^k	Availability time of machine h in stage k
α	Weight attributed to the producer
β	Weight attributed to the customer

LB_{ih}^{k} Desired lower bound for the minimum number of jobs assigned to any batch of group i on machine h in stage k

Decision variables

Z_{ijhr}^{k}	1 if job j of group i is scheduled in r^{th} position of machine h in stage k; 0 otherwise
X_{ij}^{k}	The completion time of job j of group i in stage k
T_{ij}	The tardiness of job j of group i

Mathematical formulation

$$Min\,Z = \alpha \sum_{i\in G}\sum_{j\in G_i} w_{ij}X_{ij}^{st_{ij(m_{ij})}} + \beta \sum_{i\in G}\sum_{j\in G_i} w_{ij}TD_{ij} \tag{1}$$

The objective function (1) is to *simultaneously* minimize the total weighted completion time and total weighted tardiness. Set of constraints (2) through (8), known as assignment constraint sets, are incorporated into the model to determine the optimal job sequence on machines. It is assumed that there is at most $\sum_{i\in I^k} n_i^k$ positions on each machine in stage k, so that each job of each group must be assigned to one and only one position r ($r \in N^k, N^k = \{1,2,\dots,\sum_{i\in I^k} n_i^k\}$) of a machine.

$$\sum_{h\in V_{ij}^k}\sum_{r\in N^k} Z_{ijhr}^{k} = 1 \qquad \forall i \in I^k; j \in J_i^k; k \in K; \tag{2}$$

$$\sum_{i\in I^k}\sum_{j\in J_i^k} Z_{ijhr}^{k} \geq \sum_{i\in I^k}\sum_{j\in J_i^k} Z_{ijh(r+1)}^{k} \qquad \forall r \in N^k; k \in K; h \in V^k; \tag{3}$$

$$\sum_{i\in I^k}\sum_{j\in J_i^k} Z_{ijhr}^{k} \leq 1 \qquad \forall r \in N^k; k \in K; h \in V^k; \tag{4}$$

Constraint (2) ensures that each job is assigned to only one position on machines in each stage, while constraint (3) guarantees that the jobs should be assigned to machines from the first position ($r = 1$) as consecutive positions. Since there are parallel machines in some stages, constraint (4) indicates that all positions of a machine might not be assigned to jobs.

$$X_{ij}^{k} \geq \sum_{h \in V_{ij}^{k}} \left(a_{h}^{k} + S_{0ih}^{k} + t_{ijh}^{k}\right) Z_{ijh1}^{k} \tag{5}$$

$$\forall i \in I^{k}; j \in J_{i}^{k}; k \in K;$$

$$X_{ij}^{st_{ij(1)}} \geq r_{ij} + \sum\nolimits_{h \in V_{ij}^{st_{ij(1)}}} \sum\nolimits_{r \in N^{st_{ij(1)}}} \left(t_{ijh}^{st_{ij(1)}} \times Z_{ijhr}^{k}\right) \tag{6}$$

$$\forall i \in G; j \in G_{i};$$

$$X_{ij}^{k} + M\left(1 - Z_{ijhr}^{k}\right) + M\left(1 - Z_{pqh(r-1)}^{k}\right) \geq X_{pq}^{k} + S_{pih}^{k} + t_{ijh}^{k} \tag{7}$$

$$\forall r \in N^{k} - \{1\}; i, p \in I^{k}; j \in J_{i}^{k}; q \in J_{p}^{k}; k \in K; h \in V_{ij}^{k} \cap V_{pq}^{k};$$

Set of constraints (5) through (7) determine the completion time of jobs on machines. Constraint (5) together with constraint (6) account for dynamic machine availability and dynamic job release time, respectively, while constraint (7) determines the completion time of jobs.

$$X_{ij}^{st_{ij(l)}} - X_{ij}^{st_{ij(l-1)}} \geq \sum\nolimits_{h \in V_{ij}^{st_{ij(l)}}} \sum\nolimits_{r \in N^{st_{ij(l)}}} \left(t_{ijh}^{st_{ij(l)}} \times Z_{ijhr}^{st_{ij(l)}}\right) \tag{8}$$

$$\forall i \in G; j \in G_{i}; l \in \left\{2, 3, \ldots, m_{ij}\right\};$$

The linking constraint (8) ensures the connection between completion times of a job related to each of two sequential stages, where the job had operations.

$$TD_{ij} \geq X_{ij}^{st_{ij\left(m_{ij}\right)}} - d_{ij} \tag{9}$$

$$\forall i \in G; j \in G_{i};$$

Constraint (9) is applied for finding the tardiness of each job.

$$X_{ij}^{k}, TD_{ij} \geq 0; Z_{ijhr}^{k} \in \{0, 1\} \tag{10}$$

$$\forall i \in I^{k}; j \in J_{i}^{k}; r \in N^{k}; k \in K; h \in V^{k}.$$

Finally, constraint (10) defines the variables used.