Complex Job Shop Scheduling: Formulations, Algorithms and a Healthcare Application

Thesis

presented to the Faculty of Economics and Social Sciences
at the University of Fribourg (Switzerland)
in fulfillment of the requirements for the degree of
Doctor of Economics and Social Sciences

by

Dinh Nguyen PHAM

from Vietnam

Accepted by the Faculty of Economics and Social Sciences
on February 19, 2008
at the proposal of
Prof. Dr. Heinz Gröflin (First Advisor)
and
Prof. Dr. Marino Widmer (Second Advisor)

Fribourg, Switzerland
2008

Abstract

This dissertation studies complex job shop scheduling problems. It consists of three coherent parts: (1) a study of practical features of job shop scheduling, (2) a study of a novel job shop extension problem that simultaneously addresses several practical features and its solution methods, and (3) a study of surgical case scheduling in hospitals and ambulatory surgical centers.

A job shop problem in practice often possesses one or several complexifying features that are not addressed in the classical job shop (JS) model. In the first part of this dissertation, various practical features of job shop scheduling are systematically analyzed. In addition, we formulate a number of job shop problems extended with a single additional feature as mixed integer linear programming (MILP) problems, which can be used as building blocks to formulate more complex job shop problems. To evaluate the practical use of the exact method with MILP formulation and CPLEX solver as a representative for general-purpose solvers for solving different job shop related problems, we carry out comprehensive computational experiments on benchmark data.

In the second part of the dissertation, a new job shop extension is proposed, which integrates into the JS four complexifying features namely processor flexibility, blocking constraints, sequence-dependent setup times, and job transfer times. This extension is called the Flexible Generalized Blocking Job Shop problem (FGBJS), which is capable of modeling many complex real world scheduling problems but very difficult to solve. To develop solution methods for the FGBJS, we first formulate the problem as a MILP, then solve the problem heuristically since the performance of the exact method by a MILP formulation and CPLEX solver is not good. We propose a graph representation for the FGBJS and develop three neighbor structures allowing to generate neighbor solutions by moving an operation in time and from a processor to another. Using these neighborhoods and job insertion principles, we develop three constructive heuristics. To improve a given FGBJS solution, we propose six Tabu search (TS) heuristics that result from combining the three neighborhood structures and three generic TS algorithms. Extensive computational results of these constructive and TS heuristics on 160 newly created FGBJS instances are given, which can be used to benchmark other solution methods for the FGBJS in the future.

The last part of the dissertation tackles surgical case scheduling (SCS). This is an important problem in healthcare management because of the financial impact of surgical activities in hospitals and ambulatory surgical centers (ASC). We analyze the patient flow, model the problem accordingly as a new job shop extension called the multimode blocking job shop (MMBJS), and give a MILP formulation for the MMBJS. While developing efficient solution methods for the MMBJS is still in progress, we can show that it is possible to model the SCS in ASC as a FGBJS because of several particular operational features. Further, we identify certain conditions upon which the SCS in hospitals can also be modeled as the FGBJS. In these cases, one can apply the solution methods developed in the second part of the dissertation to solve the SCS.

Acknowledgements

I am deeply grateful to Prof. Dr. Heinz Gröflin, my dissertation advisor, for his valuable guidance, kind support, and continuous encouragement throughout my work.

I have special thanks to Prof. Dr. Marino Widmer, whose constructive comments have helped to improve the quality of this dissertation.

I am also grateful to the support of Dr. Andreas Klinkert and PD Dr. Tony Hürlimann. Dr. Klinkert's works have set the base for me to expand and develop, and conversations with him have given me many insights. Dr. Hürlimann has been very supportive in consulting me on using LPL, a powerful modeling language that he has been developing for the past two decades.

I am always indebted to my wife, Mrs. Mai Thai. Her eternal love and my god-sent children's smiles are better than any thing that I can imagine of to get through hard times.

My parents and sister have always supported me even though they are thousands kilometers away from where I live. For my dad, it would be like he has finished his own dissertation, which was left undone against his wish due to political turmoil.

To my dear wife and children,
my sources of motivation.

Contents

Abstract iii

Acknowledgements v

1 Introduction 1

1.1 Preliminaries . . . 1
1.2 Examples of complex scheduling problems . . . 2
1.3 Research focus . . . 5
1.4 Dissertation's structure . . . 6

2 Job shop scheduling in practice 7

2.1 Introduction . . . 7
2.2 Problem statement and assumptions of the JS . . . 8
2.2.1 Problem statement . . . 8
2.2.2 Assumptions . . . 10
2.3 Practical scheduling features unaddressed in the JS . . . 11
2.3.1 Sequence-dependent setup times . . . 11
2.3.2 Multiple processors . . . 13
2.3.3 Job release times . . . 14
2.3.4 Transport and transfer times . . . 14
2.3.5 Processor time windows . . . 14
2.3.6 Limited buffers and blocking constraints . . . 15
2.3.7 No-wait constraints . . . 16
2.3.8 Generalized no-wait constraints . . . 17
2.3.9 Processor flexibility . . . 18
2.3.10 Objective functions . . . 20
2.4 Evaluation of the JS' MILP formulations . . . 21
2.4.1 Manne formulation . . . 23

2.4.2 Liao-You formulation . . . 23
2.4.3 Adaptive Manne formulation . . . 24
2.4.4 Wagner formulation . . . 24
2.4.5 Wilson formulation . . . 26
2.4.6 Evaluation of the JS' MILP formulations by a general-purpose solver . . . 26
2.5 Formulations of practical job shop scheduling problems . . . 30
2.5.1 Formulations of 1-feature JS extension problems . . . 30
2.5.2 Formulation of complex job shop scheduling problems . . . 37
2.5.3 Evaluation of the mathematical programming approach by a general-purpose solver . . . 40
2.6 Overview of other solution approaches to job shop related problems . . . 43
2.6.1 Exact methods . . . 43
2.6.2 Constructive heuristics . . . 44
2.6.3 Iterative heuristics . . . 45
2.7 Concluding remarks . . . 46

3 Flexible generalized blocking job shop problem 47
3.1 Introduction . . . 47
3.2 The FGBJS and some applications . . . 48
3.3 Problem formulation . . . 50
3.4 The Generalized Blocking Job Shop . . . 54
3.4.1 Graph representation for the GBJS . . . 54
3.4.2 Job insertion and conflict graph . . . 56
3.4.3 Feasible schedule generation . . . 61
3.5 Graph representation for the FGBJS . . . 63
3.6 Neighborhood structures for the FGBJS . . . 68
3.6.1 Simple neighborhood $N^s(x)$ . . . 69
3.6.2 Pairwise exchange neighborhood $N^p(x)$ . . . 74
3.6.3 Flexible closure neighborhood $N^c(x)$ . . . 74
3.6.4 Move representations . . . 78
3.7 Constructive heuristics . . . 78
3.7.1 A general framework for constructive heuristics . . . 78
3.7.2 Most critical operation heuristic . . . 80
3.7.3 Most favorable position heuristic . . . 82
3.7.4 Bottleneck heuristic . . . 85
3.8 Tabu search algorithms . . . 86

3.8.1 Three generic Tabu search algorithms . . . 87
3.8.2 Tabu search algorithms for the FGBJS . . . 93
3.9 Computational results . . . 94
3.9.1 Benchmark instances . . . 94
3.9.2 Computational results on the constructive heuristics . . . 95
3.9.3 Computational results of the Tabu search heuristics . . . 99
3.10 Concluding remarks . . . 115

4 Application in surgical case scheduling **117**
4.1 Introduction . . . 117
4.2 Problem descriptions . . . 118
4.3 Literature review . . . 120
4.4 MILP model for SCS . . . 123
4.4.1 Multi-mode blocking job shop problem . . . 123
4.4.2 MILP model for SCS based on the MMBJS . . . 124
4.4.3 Discussion . . . 128
4.4.4 Computational experiments with the SCS' MILP formulation . . . 133
4.5 Heuristic approach to the SCS problem . . . 136
4.5.1 Decomposition procedure for the weekly SCS problem . . . 136
4.5.2 Heuristic approach to the daily SCS problem in ACS . . . 138
4.5.3 Heuristic approach to the daily SCS problem in integrated hospitals . . . 140
4.6 Concluding remarks . . . 142

5 Conclusions **145**

Bibliography **149**

Appendices **157**
.1 Scheduling terminologies . . . 157
.2 Subroutine algorithms in Chapter 3 . . . 159
.3 Performance of the FGBJS' MILP formulation . . . 161

List of Tables

2-1 Processing times in Example 2.1. 9
2-2 Sequence-dependent setup times in Example 2.2. 13
2-3 Operations overlapping pattern of Example 2.7. 18
2-4 Processing times in Example 2.8. 19
2-5 Notations for the JS formulations 22
2-6 Size complexity of the JS' MILP formulations. 27
2-7 Performance of the JS formulations. 29
2-8 Summary of performance of the JS formulations. 30
2-9 Processors in modes. 39
2-10 Mode-dependent processing times. 39
2-11 Performance of the MILP approach for on instances of the BJS and the NWJS. . 41
2-12 Performance of the MILP approach on instances of the FJS. 42
3-1 Processing times in Example 3.1 51
3-2 Sequence dependent setup times in Example 3.1. 51
3-3 Notations for the MILP. 52
3-4 Simple moves in Example 3.9. 73
3-5 Flexibility levels in FJS benchmark instances. 94
3-6 Performance of the constructive heuristics on *sdata* set. 96
3-7 Performance of the constructive heuristics on *edata* set. 96
3-8 Performance of the constructive heuristics on *rdata* set. 97
3-9 Performance of the constructive heuristics on *vdata* set. 97
3-10 Performance of the constructive heuristics w.r.t. improvement to permutation schedules. 98
3-11 Performance of the constructive heuristics w.r.t. deviations. 98
3-12 Performance of the constructive heuristics w.r.t. numbers of best makespans obtained. 98
3-13 Computing times of the constructive heuristics. 99
3-14 Impact of flexibility on performance of the constructive heuristics. 101

3-15 Impact of flexibility on the number of best makespans obtained. 101
3-16 Impact of instance size on the performance of the constructive heuristics. 102
3-17 Ranks of the TS algorithms with respect to flexibility levels. 102
3-18 Ranks of the TS algorithms with respect to instance sizes. 103
3-19 Performance of the TS algorithms on *sdata* set. 104
3-20 Performance of the TS algorithms on *edata* set. 105
3-21 Performance of the TS algorithms on *rdata* set. 106
3-22 Performance of the TS algorithms on *vdata* set. 107
3-23 Ranks of the generic TS algorithms w.r.t. flexibility levels. 108
3-24 Ranks of the generic Tabu search algorithms w.r.t. instance sizes. 110
3-25 Nonconformity with SG-MFP and SG-MCO as initial schedule generators. 111
3-26 Nonconformity with SG-MFP and SG-BN as initial schedule generators. 112
3-27 Nonconformity with SG-MCO and SG-BN as initial schedule generators. 113
3-28 Nonconformity with permutation schedules as starting solutions. 114
4-1 Example of OR block allocation table. 120
4-2 Causes of OR wasted time 122
4-3 Notations for the MMBJS's MILP formulation. 126
4-4 OR block allocation table in Example 4.1. 133
4-5 Modes and their availability interval of Example 4.1. 134
4-6 Surgical cases with modes and processing times of Example 4.1. 134
4-7 Optimal solution with operations' modes, starting time, and leaving time of Example 4.1. 135
4-8 Results of nine SCS test instances. 136
4-9 Results of four ASC instances modeled as the FGBJS. 139
4-10 Processors and their availability interval of Example 4.3. 141
4-11 Surgical cases with processors and processing times in Example 4.3. 142
-1 Performance of the FGBJS's MILP formulation with CPLEX 9.0. 161
-2 MILP formulation with CPLEX 9.0 vs. heuristics for *sdata* set. 162
-3 MILP formulation with CPLEX 9.0 vs. heuristics for *edata* set. 162
-4 MILP formulation with CPLEX 9.0 vs. heuristics for *rdata* set. 162
-5 MILP formulation with CPLEX 9.0 vs. heuristics for *vdata* set. 162

List of Figures

1-1 A robot cell. 2
1-2 An automated high-density warehouse. 3
1-3 Patient flow for surgical cases. 4
1-4 A railway system with block sections. 4
2-1 Gantt chart of the JS in Example 2.1. 9
2-2 Disjunctive graph and feasible selection of Example 2.1. 10
2-3 Gantt charts for a job shop with sequence-dependent setup times. 13
2-4 Gantt charts for a job shop with multiprocessor requirement. 14
2-5 Gantt chart for a job shop with processors' time windows. 15
2-6 Gantt chart for a job shop with blocking constraints. 16
2-7 Gantt chart for a job shop with the no-wait constraint. 17
2-8 Gantt charts for a job shop with generalized no-wait constraints. 18
2-9 Gantt chart for a job shop with processor flexibility. 19
2-10 Modeling the JS with limited buffers as the BJS. 34
2-11 Gantt chart for Example 2.9. 39
2-12 Structure tree for complex job shop scheduling problems. 40
3-1 Gantt chart in Example 3.1. 51
3-2 Disjunctive blocking graph for Example 3.2. 56
3-3 A short cycle visiting job J once. 58
3-4 Insertion graph G^2 and its associated conflict graph H_{G^2} in Example 3.3. 59
3-5 Closure in Example 3.4. 60
3-6 A feasible selection in G^2. 62
3-7 Disjunctive blocking graphs w.r.t. operation 3. 65
3-8 A feasible selection for Example 3.8. 66
3-9 Two consecutive operations of a job are on are the same processor. 67
3-10 Deadlock situation involving two jobs. 67
3-11 Critical path, nodes, arcs, and operations. 69

3-12 Simple move. 70
3-13 Simple move in between two operations involved in a 3-job swap. 70
3-14 Infeasible insertions to the left and right. 72
3-15 Simple neighbor for Example 3.9 73
3-16 A feasible job selection in G^J or a stable set of maximum cardinality in H_{G^J}. . . 76
3-17 Flexible closure move of Example 3.10. 77
3-18 Job blocks in a critical path. 77
3-19 Feasible schedule obtained by the SG-MOC for Example 3.12 82
3-20 Feasible schedule obtained by the SG-MFP for Example 3.13. 84
3-21 Feasible schedule obtained by the SG-BN for Example 3.14. 86
4-1 Patient flows in integrated hospitals. 119
4-2 Patient flows in ambulatory surgical centers. 119
4-3 Single availability interval modes 129
4-4 Gantt chart of Example 4.1. 135
4-5 Gantt chart after an emergency arrival in day 1 in Example 4.1. 136
4-6 Gantt charts in Example 4.3. 142
-1 Example of using Gantt chart to represent a schedule. 157
-2 Example of local left-shifts. 158
-3 Example of global left-shifts. 158

Chapter 1

Introduction

1.1 Preliminaries

Suppose there are a number of jobs to be processed by several processors. Such processors can be machines in a manufacturing plant or doctors in a hospital, while jobs can be customer orders in a manufacturing plant or patient check-ups in a hospital. Processing of a job by a processor is called an *operation*. *Scheduling* is the process of allocating the processors to the operations over time. The result of scheduling is a *schedule*, which assigns some processor(s) to each operation during a specific time period. In other words, a schedule is a plan that tells us "the detailed timing of the operations within the capability of the resources" [Bak74]. Examples of schedules are bus timetables, course time plans, operating room schedules, production schedules, etc. A schedule is said to be *feasible* if it satisfies certain constraints in the context where scheduling decisions are made. Two basic constraints are: (1) technology precedence constraints which demand that certain operations be performed in some specific order, and (2) capacity constraints which require that the workload imposed on a processor should not exceed its capacity at any time. A *scheduling problem* is a problem of determining a feasible schedule for given operations and processors in order to meet some objective set by the management. Solving a scheduling problem is to seek answers to the following two questions [Bak74]:

1. Which processors will be allocated to perform each operation?

2. When will each operation be performed?

Thus, solving a scheduling problem is about making decisions on assigning processor to operations and sequencing operations at the same time.

When processors are scarce and hence required by competing operations, scheduling problems can be very complex. The next section introduces some examples of such complex scheduling problems that are to be referred to throughout this dissertation.

1.2 Examples of complex scheduling problems

Example 1.1. *A manufacturing scheduling problem*

An automated manufacturing system can have several robot cells. A typical robot cell is comprised of several Computer Numerical Controlled (CNC) machines and a transport robot. These CNC machines process some parts while the robot takes care of all material handling tasks. A real-life example of robot cell with three CNC machines described in [RJ96] is shown in Figure 1-1.

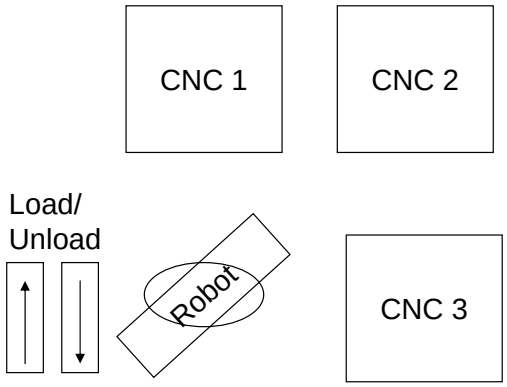

Figure 1-1: A robot cell.

Each part has its own sequence of operations. Each CNC machine can be configured with appropriate tools to process more than one part type. The robot takes a part from the load/unload area, feeds the part to a preassigned machine, makes an empty move to another machine to take a part waiting there, and moves it to the next machine to process. If the robot holding a part reaches the machine on which the next operation of the part takes place and the machine is still processing another part, it cannot discharge the part due to lack of immediate inventory buffer and therefore, it is blocked by the part and cannot perform the next task. On the other hand, if a machine has finished a part and the robot is not available to take the part from the machine, the machine cannot process another part and is blocked with the waiting part. The shop floor manager's task is to decide which machine processes which part in which sequence and in which sequence the robot transports the parts so that all parts are finished in the shortest time possible.

Example 1.2. *A logistic scheduling problem*

An automated high-density warehouse that stores a very large number of pallets is studied in [GK05]. The warehouse is divided into several floors that are linked by elevators. Each floor is divided into corridors where pallets are stored. Cross-aisles perpendicular to the corridors

connect them to the elevators. Each corridor is equipped with one carriage and each cross-aisle has one cross-aisle carriage (see Figure 1-2). Pallets to be retrieved or stored daily in the

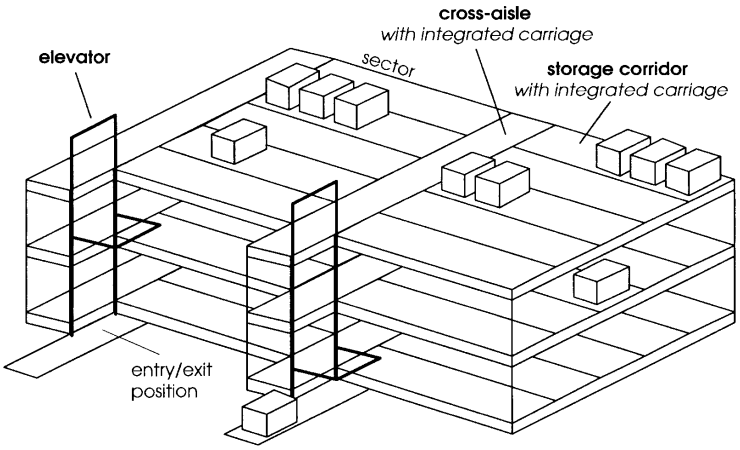

Figure 1-2: An automated high-density warehouse.

warehouse are known in advance. The retrieval of a pallet consists of a sequence of operations. First, the carriage of the corridor where the pallet is stored makes an empty move to the pallet, takes the pallet, and moves it to the adjacent cross-aisle. The corridor carriage then waits for the cross-aisle's carriage to come to hand over the pallet. Upon its arrival, the cross-aisle carriage takes over the pallet and carries it to an elevator. If the elevator is not yet available, the cross-aisle carriage has to wait while holding the pallet. When the elevator arrives, it takes the pallet and moves it down to the loading area, where the pallet is loaded into a truck and driven away. After each pallet transfer, the empty corridor or cross-aisle carriage or elevator makes a move to another position to handle another pallet. The pallet storage process is done in the reversed way. The warehouse manager, respectively the control logic, should decide how to retrieve and store the pallets in order to maximize the daily throughput of the warehouse.

Example 1.3. *A service scheduling problem*

A health care facility (a hospital or an ambulatory surgical center) performs tens of surgical operations per day in several operating room suites. When a patient arrives at the facility for an operation, he or she has to go through three phases: preoperative, perioperative, and post-operative phases. First, the patient is prepared in a holding unit. Then the patient undergoes an operation performed in an appropriate operating room by a surgical team consisting of one main surgeon, possibly one assistant surgeon, one anesthetist, and several nurses. After the operation is finished, the patient is moved to a recovering room and stays there for some time before being either discharged or admitted to a nursing ward (see Figure 1-3).

Operating rooms are very expensive to run, especially in overtime. If a case is cancelled because of lack of necessary resources, the patient will be negatively impacted with respect to his or her

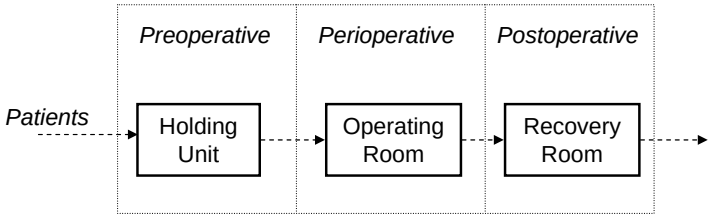

Figure 1-3: Patient flow for surgical cases.

health status and emotional well-being. Operating room managers need to schedule the cases in a safe way for the patients while making efficient use of the facility's resources.

Example 1.4. *A transport scheduling problem* [1]

A railway scheduling problem is reported by D'Ariano, Pacciarelli and Pranzo [DPP06] as follows. A railway network consists basically of track segments and control signals. Each train goes through a number of specified track segments and may dwell in some stations along its route, each for a certain amount of time. There are signals to control the enter and exit of any train before every track junction, along the lines for every specified distance, and inside the stations. A track segment between two control signals is called a *block section*. Only one train can occupy a block section at a time. In the graph below, there are two junctions and nine block sections. A collision between train T1 and T2 can take place in block sections 5 and 6.

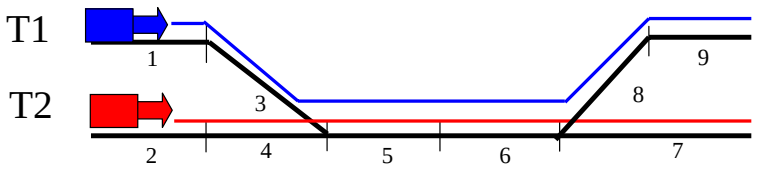

Figure 1-4: A railway system with block sections.

Block signal can be either red (R) or yellow (Y) or green (G). The red control signal indicates that the next block section is either out of service or still occupied and hence a train entering a section with red signal should stop completely by the end of this section. The yellow signal means that the next block section is free, but the block after next is occupied, so the train can only enter the next block section at a limited speed. If the signal is green, the two next consecutive blocks sections are free and the train can continue its travel at full speed. The running time of a train in a block section is counted from the time its head enters the section. After the whole train leaves this block section, the block's control signal before the previous

[1] Special thanks go to Prof. Dr. Peter Brucker for his communication note which further explained the operations issues in railway scheduling.

section turns yellow and the one before that switches to green. For example, signal states when a train runs through a railway segment consisting of four sections (initially free) are as follows: (1) GGGG, (2) **R**GGG, (3) R**R**GG, (4)YR**R**G, (5) GYR**R**, (7) GGYR, (8) GGGY, and (9) GGGG, where a signal (R,Y or G) in bold indicates that the train is in the signal's associated section.

The timetable for trains in the railway network is usually established in advance with detailed information on the routing and starting times for each train. However, disturbances from various causes can delay some trains. Since each train occupies a railway block section at a time, a delayed train could collide with another train that enters the section as previously scheduled. Therefore, a vital task for railway schedulers is to correct in real-time the starting time of each train in each of its block sections after some delay has occurred. This problem is termed *conflict resolution problem* (CRP). Solving the CRP aims at obtaining a feasible solution that ensures no collision and minimizes so-called *secondary delay*, which is the difference between the actual arriving time of a train and its latest arriving time as previously scheduled.

1.3 Research focus

In order to survive and grow in today's business which is characterized by fierce competition and high expectations from customers, any business entity should try to use its existing resources as efficiently as possible to effectively serve its customers. Making sound scheduling decisions can contribute substantially to the successful use of resources of a company. "Interfaces", the INFORMS' flagship practitioner-oriented journal, has published several success stories where companies use scheduling systems to gain competitive advantages. Nevertheless, the application of scheduling theory to practice is still limited [PR00]. The gap between scheduling theory and practical needs appears significant as remarked by Dudek, Panwalker, and Smith (1992): "At this time, it appears that one research paper (that of Johnson, 1954 [Joh54]) set a wave of research in motion that devoured scores of person-years research time to an intractable problem that is of little practical consequence " [DPS92].

This dissertation is set to make a contributive step in bridging scheduling theory and the practical world by working on three "complementary parts":

1. *Review of practical job shop scheduling features*

   Basically, a job shop is a shop environment where job processing goes through multiple stages and routings of different jobs may be different. It is the most encountered shop environment in industries. The first part of this dissertation focuses on identifying the features of practical job shop scheduling problems and formulating them mathematically.

2. *Development of a realistic scheduling model and its algorithmic solutions*

Based on the findings under part 1, part 2 proposes a scheduling model capable of modeling a wide range of job shop scheduling applications where practical scheduling features of blocking constraints, flexibility, and sequence-dependent setup times are present. The second task for this part is to develop good solution methods for this practical job shop scheduling model.

3. *Application*

   The last part concerns an application area of Operations Research that is finding growing interest, namely healthcare services. Specifically, it addresses scheduling of surgical operations, using methods and findings from the two preceding parts.

1.4 Dissertation's structure

The dissertation is organized as follows. Chapter 2 analyzes various features of scheduling problems as they occur in practice. Then each of these features is integrated into a base model called *Job Shop* (JS) to form the corresponding single-feature extension of the JS. Each JS extension is then formulated as a mixed integer linear programming problem (MILP) from a base formulation selected from several existing formulations for the JS.

Chapters 3 proposes a new complex job shop scheduling model that has not yet been addressed in the literature, although it is of practical relevance. This model, which we call the *Flexible Generalized Blocking Job Shop* , integrates simultaneously into the JS several features mentioned in Chapter 2, namely sequence-dependent setup times, job transfer times, blocking constraints, and processor flexibility. The model is shown to be capable of modeling many practical scheduling problems, including all of the example problems mentioned in Section 1.2. As the problem is too difficult to be solved optimally, several heuristics based on Tabu search approach are proposed together with three constructive heuristics, which are to find feasible starting solutions.

Chapter 4 is devoted to the application part. The targeted application area is surgical case scheduling, which is one of the pertinent issues in healthcare management. This chapter presents a generalized job shop scheduling model and its MILP formulation. Since the model is too complex to solve by exact algorithms, a scheme is proposed to heuristically solve the surgical case scheduling problem with the help of the model and solution methods developed in Chapter 3.

Chapter 5 concludes this dissertation with a summary of contributions and a future research outline. Several scheduling terminologies and subroutine algorithms are found in the Appendix.

Chapter 2

Job shop scheduling in practice

2.1 Introduction

Chapter 1 describes several practical scheduling problems in various industrial sectors, from manufacturing to services. The shop environments of these problems have one thing in common, which could be termed the job shop environment with multistage job processing and nonidentical job routings. There is a standard scheduling model in scheduling literature called *job shop (JS)* that deals with scheduling in a job shop environment. In the JS, a given number of jobs have to be executed by some processors. All jobs and processors are available from time zero. Each job consists of a chain of operations to be processed in that order. Each operation is performed by a preassigned processor without preemption. The processing time of any operation is deterministic, independent of other processing times, and known in advance. Each operation's transfer, transport, and setup times (if any) are included in its processing time. There are unlimited buffers before and after any processor to hold in-process jobs. Each processor can process only one job at a time. Any feasible schedule should determine the starting time for each operation such that there are no operations overlapping in any job's chain of operations and in any processor's operation sequence. The objective is to find a feasible schedule that minimizes the time to finish all the jobs, which is commonly referred to as *makespan*.

As the JS model does not cover many real-life scheduling aspects, applying it directly to solve a practical problem might result in solving a "wrong" problem [MSB88][WR90][Sch98]. For this reason, many research works have been focusing on extending the JS model for different industrial settings. One of the common approaches to attack an extended JS is mathematical programming, which formulates the JS extension as a mixed integer linear programming (MILP) problem and then tries to solve the formulated problem by a general-purpose mixed integer linear programming solver. According to Rinnooy Kan [RK76], it is a *natural* solution approach to scheduling problems. Following this research stream, this chapter presents a number of real-life job shop scheduling features along with their corresponding MILP formulations. The practical

features introduced in this chapter are not new but some of them, to our knowledge, have not been formally formulated in the scheduling literature. By gathering various job shop related formulations in a single reference, this chapter complements earlier formulation compiling works which do not sufficiently cover the JS [BDW99] or its extensions [Pan97].

This chapter is organized in seven sections. Section 2.2 formally defines the JS problem and classifies the problem's assumptions. Section 2.3 outlines several practical features of scheduling, namely sequence-dependence setup times, multiprocessor requirements, job release times, limited buffers, blocking constraints, no-wait and generalized no-wait constraints, transport and transfer times, processor time windows, and processor flexibility. All of these features, except the last one, are of constraining nature. Different objective functions other than the makespan are also discussed. Section 2.4 reviews and evaluates the existing mathematical programming formulations for the JS. Section 2.5 presents MILP formulations for 1-feature extensions of the JS with respect to the features introduced in Section 2.4 and demonstrates through an illustrative example how to formulate a complex scheduling problem with multiple features based on several of the developed 1-feature formulations. Section 2.6 reports on the computational experiments on various benchmark instances to evaluate the mathematical programming approach and gives a brief overview on other solution methods to job shop related problems. Concluding remarks are given in Section 2.7.

2.2 Problem statement and assumptions of the JS

2.2.1 Problem statement

The classical Job Shop Scheduling problem, in short JS, can be stated formally as follows. Let $\mathcal{J}$ be a set of n jobs, M a set of m processors, and I be a set of o operations. A job $J \in \mathcal{J}$ is a set of operations to be performed in a fixed order. Therefore, $J \subseteq I$ and $\mathcal{J}$ is a partition of I, i.e. $\mathcal{J} \subseteq 2^I$. For any job J, index its operations in the processing order as $J_1, \ldots, J_{|J|}$. Define the set of pairs of consecutive operations for job J as $A_J = \{(J_r, J_{r+1}) : 1 \leq r < |J|\}$. For each $(i,j) \in A_J$, i is the immediate job predecessor of j, $i = JP(j)$, and j is the immediate job successor of i, $j = JS(i)$. Define σ and τ as two dummy operations of zero duration, σ starts before and τ starts after every operation. Each operation $i \in I$ is performed by a preassigned processor $\mu(i) \in M$ for a known and deterministic duration $p_i > 0$. Denote by I_k the set of all operations to be performed by processor k and by B the conflict set of all *unordered* pairs (i,j) of operations i and j performed on the same processor, i.e. $B = \{(i,j) : i,j \in I_k, i \neq j, k \in M\}$. Each processor can process only one operation at a time. All jobs and processors are continuously available from time zero. No preemption is allowed for any operation. The objective is to find a feasible schedule that respects the processing order of each job and the capacity of each processor and minimizes the makespan. According to Graham et al.'s 3-field classifications [GLLRK79], the JS can be addressed as $J||C_{\max}$.

Example 2.1.

Consider a JS instance of 3 jobs $\mathcal{J} = \{1, 2, 3\}$, 3 processors $M = \{1, 2, 3\}$, and 9 operations $I = \{1, 2, ..., 9\}$. The operations are partitioned into jobs and listed in the jobs' processing order as follows: job 1 $= \{1, 2, 3\}$, job 2 $= \{4, 5, 6\}$, and job 3 $= \{7, 8, 9\}$. The assigned processor and processing time for each operation are given in Table 2-1. The set of pairs of consecutive

Proc/Op	1	2	3	4	5	6	7	8	9
1	10				20				30
2		20		10				20	
3			30			30	10		

Table 2-1: Processing times in Example 2.1.

operations for the jobs are $A_1 = \{(1, 2), (2, 3)\}, A_2 = \{(4, 5), (5, 6)\}$, and $A_3 = \{(7, 8), (8, 9)\}$. The operation sets for each processor are $I_1 = \{1, 5, 9\}, I_2 = \{2, 4, 8\}$ and $I_3 = \{3, 6, 7\}$. The conflict set is $B = \{(1, 5), (1, 9), (5, 9), (2, 4), (2, 8), (4, 8), (3, 6), (3, 7), (6, 7)\}$.

Figure 2-1 shows the Gantt chart of a feasible solution of makespan 90. It is easy to see that this solution is non-delay (see Appendix for the concept of non-delay schedules). □

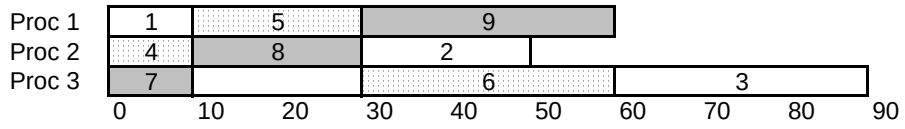

Figure 2-1: Gantt chart of the JS in Example 2.1.

The JS is often represented graphically by a so-called disjunctive graph [ABZ88][1]. This is a arc-weighted graph $G = (N, A \cup A^{\sigma,\tau}, E)$, where node set N contains for each operation $i \in I \cup \{\sigma, \tau\}$ a representative node $i \in N$. Conjunctive arc set A contains for each pair of consecutive operations $i, j, (i, j) \in A_J, J \in \mathcal{J}$ a conjunctive arc $(i, j) \in A$ having a weight equal to the processing time of operation i. The arc set $A^{\sigma,\tau}$comprises zero-weighted arcs joining the node representing dummy start operation σ with the nodes associated with the first operation of each job and positively weighted arcs linking the nodes associated with the last operation of each job with the node representing dummy end operation τ, these arcs' weight equals the corresponding operations' processing time. The disjunctive arc set E contains for each pair (i, j), $i, j \in I_k$ a pair of two disjunctive arcs $\{(i, j), (j, i)\}$ having respective weights p_i and p_j, the former arc implies i precedes j on k and the latter implies the reverse order. A selection S of disjunctive arcs in G, $S \subseteq E$, is feasible if $S \cap \{(i, j), (j, i)\} = 1$ for every pair of disjunctive arcs and the resulting solution graph $G(S) = (V, A \cup S)$ is acyclic. The JS is equivalent to the problem of making a feasible selection S so that the length of the longest path from node σ to node τ (the critical path) in $G(S)$ is minimized. Each feasible selection then corresponds to a

[1]The disjunctive graph initially proposed by Roy and Sussmann in 1964 [RS64] is node-weighted.

unique semi-active schedule (see Appendix for the concept of semi-active schedules) in which the earliest starting time of operation i equals the length of the longest path (σ, i). Conversely, from any semi-active schedule we can easily construct its corresponding feasible selection.

The disjunctive graph for Example 2.1 below has conjunctive arcs drawn in solid arcs and pairs of disjunctive arcs in non-weighted edges for the sake of simplicity. The selection corresponding to the solution presented in Figure 2-1 is displayed to the right of Figure 2-2.

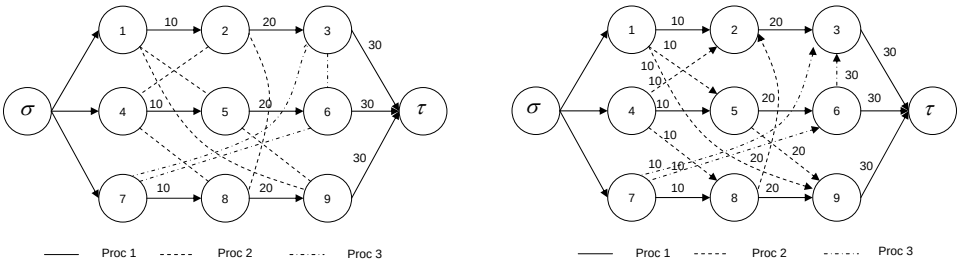

Figure 2-2: Disjunctive graph and feasible selection of Example 2.1.

2.2.2 Assumptions

The assumptions made in the JS can be classified into the following groups.

1. Assumptions concerning jobs

   J1. Each job is released at the beginning of the scheduling period and available to be processed at any time.

   J2. There is a precedence relation between any two operations of a same job but not between any two operations of different jobs.

   J3. The routing for each job is defined by its operation sequence and a preassigned processor for each of its operations. The routings of all jobs may not be the same.

   J4. Each operation takes a continuous positive and deterministic processing time, which includes job transport and setup times (if any).

   J5. There is no due date for any job.

2. Assumptions concerning processors

   M1. Each processor is continuously available throughout the scheduling period without any breakdown or maintenance.

3. Assumptions concerning job processing

   P1. All jobs must be completed.

P2. Each processor processes at most one job (or equivalently, performs at most one operation) at a time.

P3. Each operation, once started, must be finished without any preemption.

P4. There are unlimited buffers before and after each processor to store in-process jobs.

P5. Each job can be processed by only one processor at a time.

P6. The processing of any job must follow its defined precedence order of operations.

P7. The time to switch between any two operations to be performed consecutively by a processor is zero.

In practice, one or several of the assumptions in the JS might not hold, depending on the particular shop environment. The next section examines this issue in detail.

2.3 Practical scheduling features unaddressed in the JS

The practical scheduling features introduced in this section can be divided into feature groups concerning jobs, processors, job processing, and goals. Belonging to the first group are the features of sequence-dependent setup times, multiprocessors, job release times, and job transport and transfer times. The second group contains only the feature of processors' time windows. The third group comprises the other features including limited buffers, blocking constraints, no-wait constraints, generalized no-wait constraints, and processor flexibility, and the last group deals with various objective functions. Often more than one of these features can be observed in a real-life job shop scheduling problem. From a methodological viewpoint, extending the JS to include these features surely complexifies the problem, which is already very difficult to solve. On the other hand, by excluding them in order to have a "clean" model for mathematical analysis, one runs the risk of further alienating scheduling research from practical needs [MSB88]. Lee, Lei, and Pinado observed that the current trend in deterministic scheduling is to take on the challenge of extending (standard scheduling) models to include practical constraints instead of retreating into the study of clean models [LdMH97].

2.3.1 Sequence-dependent setup times

Setup involves preparing or finalizing an operation. The time required to perform a setup is called setup time. In manufacturing settings, setups include obtaining proper tools, inspecting materials, fixing parts, adjusting tools, removing parts, and cleaning machines after operations. Setup activities in non-manufacturing sectors exist in various formats. For instance, material handling's setups in warehouses involve moving idle carriages to their right positions to store/retrieve the right pallet (Example 1.2); setups in surgical operations involve cleaning an operating room after an operation and preparing surgical tools and supporting appliances for

a next operation (Example 1.3); setups in scientific computing involve unloading and loading executive programs from memory, etc.

A setup is said to be *separable* if an operation's setup can be separated from its job processing, i.e. the operation's setup can done on the operation's available dedicated processor while its job predecessor is still in process and *inseparable* if otherwise. Setup times can be either *sequence-independent* or *sequence-dependent*. In the first case, a setup time to process a job on a processor depends only on the job. In the second case, the setup time depends on both the current job and its immediate preceding job on the processor. A scheduling problem with sequence-dependent setup times is obviously more general than a problem with sequence-independent setup times. Further classification distinguishes *batch setups* where setup is required only between two consecutive batches but not between two jobs of the same batch from *non-batch* setups where jobs are not grouped into batches and setup time of a job is specified according to the job only.

The JS assumes that sequence-dependent setup times are either included in operations' processing times or negligibly small (assumption J4); and there is no sequence-dependent setup between any two operations (assumption P7). These assumptions do not hold in many settings, e.g. printing industries which typically have long sequence-dependent setup times. Recognizing the presence of setup times reveals opportunities to reduce them, hence to shorten the total lead time and make the process more flexible to changes in customers' demands. Although different setup time reduction measures (e.g. single-minute-exchange-of-dies SMED techniques, or component modularization in manufacturing, or standard surgical tool kit in surgical services) have been applied very successfully [AS06], it might take a long effort for many sectors, especially in services, to reduce setup activities substantially.

Examples of separable, non-batch, and sequence-dependent setups can be found in all of the example scheduling problems in Chapter 1. In the cell manufacturing example, the setup time for the part-moving operation is the time it takes for a robot to move without any part from one CNC machine to another. Because this time is proportional to the distance between two machines, it is sequence-dependent. The transport times of empty carriages in Example 1.2 can be treated similarly. In Example 1.3, two consecutive identical cases may need the same supporting equipment while non-identical cases may require different surgical tools, thus setup times for surgical operations can be considered as sequence-dependent to some extent. In the railway scheduling problem in Example 1.4, a train can only enter a block if the previous train occupying the block leaves it completely. The duration from the time the previous train's first axle leaves the block until the time its last axle leaves the block is considered as the setup time for the train, which is determined by the previous train's length and speed and hence sequence-dependent. More examples of practical problems involving sequence-dependent setup times can be found in the latest comprehensive survey by Allahverdi et al. [ANCK06].

Example 2.2.

To illustrate the impact of sequence-dependent setup times on the makespan, consider Example 2.1 with additional sequence-dependent setup times on processor 3 as given in Table 2-2 while other setup times are zero. Observe in the upper Gantt chart in Figure 2-3 that the setup times lengthens the previous schedule's makespan from 90 to 120. A better schedule of makespan 110 (shown on the lower chart) is obtained by switching the sequence between operation 3 and 6.

From/To	3	6	7	τ
σ	10	10	10	-
3	-	10	10	10
6	10	-	10	10
7	10	20	-	10

Table 2-2: Sequence-dependent setup times in Example 2.2.

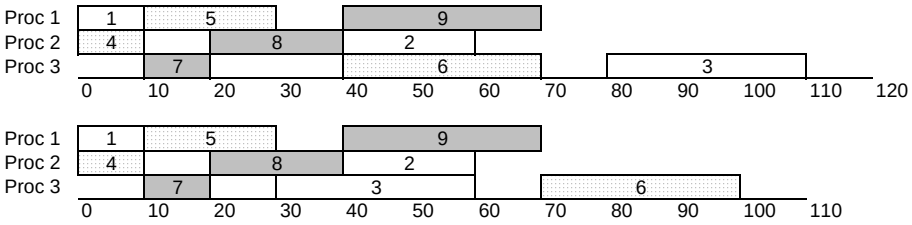

Figure 2-3: Gantt charts for a job shop with sequence-dependent setup times.

2.3.2 Multiple processors

Assumption P5 states that each operation is done by a single processor. In practice, several processors may be needed simultaneously to perform an operation. For example, a machine and an operator are required to operate a part; a surgery is done by a surgical team (Example 1.2); several processors are allocated to perform threads of a computing task; and so on. Simultaneous use of parallel resources might be used to speed up job processing. For an overview of scheduling with multiprocessors, we refer the reader to [Dro96]. According to this review, the majority of researches in scheduling with the multiprocessor feature are devoted to parallel and flow shop scheduling, not job shop scheduling.

Example 2.3.

Assume that operation 5 needs processors 1 and 2 together instead of single processor 1 as in Example 2.1, and its processing time is now half of its previous processing time on processor 1, i.e. 10 time units. The processor and processing time for each of the other operations, as well as the sequences, are not changed. The Gantt chart below shows that although time of operation 5 is performed in half the time, due to the simultaneous need of processors 1 and 2 by this operation, the resulting makespan can increase (100, see the Gantt chart below) in comparison with the initial makespan 90 in Example 2.1.

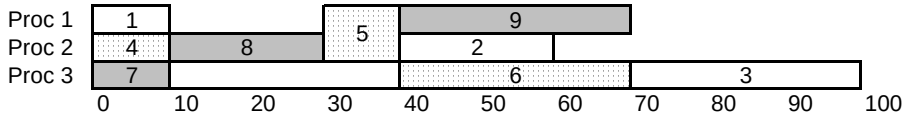

Figure 2-4: Gantt charts for a job shop with multiprocessor requirement.

2.3.3 Job release times

In the JS, all jobs are available to be processed from time zero onwards (assumption J1). Without considering dynamic job arrivals, the assumption of zero release (ready) times still does not always hold. For instance, manufacturing parts from different customers may arrive at a plant at different fixed times in a day; some surgeries are performed only after their same-day medical tests' results are made available, etc.

2.3.4 Transport and transfer times

After being processed by a processor, a job is said to be *transferred* to the next processor if both the current processor and the next processor are involved in the transfer, and it is said to be *transported* if otherwise. Two synchronized steps of a job transfer are: (1) the hand-over step where the job is handed over to the next processor and (2) the take-over step where the next processor takes over the job from the current processor. Typically the two steps have same duration. Since in general the job waits on the processor where it has been processed until the processor hands over the job to the next processor, transfers occur together with blocking constraints (to be discussed in Section 2.3.6).

Transport times are also negligible or included in processing times according to assumption J5. This assumption is not valid when the system has at least one of the following characteristics: (1) transporters are not always available; (2) processors are located remotely from one another; (3) distances among processors are not identical; and (4) speeds of transporters are not identical. When transporters are of high demand as per condition (1), they should be considered as scarce resources like other processors. Transport times are not negligible under condition (2) and they do not have a constant value under conditions (3) and (4). Our scheduling examples show that in many settings, empty moves of transporters should be viewed as sequence-dependent setup times.

2.3.5 Processor time windows

Processors in the JS are assumed to be continuously available from time zero (assumption M1). This is hardly true in practice due to machine breakdowns or planned maintenances in manufacturing industries, or personnel absences in service sectors. Processor unavailability of stochastic nature, e.g. machine breakdowns or personnel sick leaves, is out of the scope of this study.

Other processor unavailability is deterministic and known in advance, e.g. unavailability due to machine maintenance, personnel vacations, working shifts, etc. Therefore, we can associate with each processor one or several time windows during which the processor is available to process the jobs.

Example 2.4.

Suppose processor 1 is available in time window $[10, 80]$, processor 2 in $[20, 80]$, and processor 3 in $[10, 100]$. It appears that each processor's capacity is sufficient to meet its assigned workload requirement (60 for processor 1, 50 for processor 2, and 80 for processor 3), but there is no feasible solution to the problem because the operation precedence constraints reduce the capacity of processor 3. If the time window for processor 3 is extended to $[10, 110]$ then we can obtain a feasible solution as shown below.

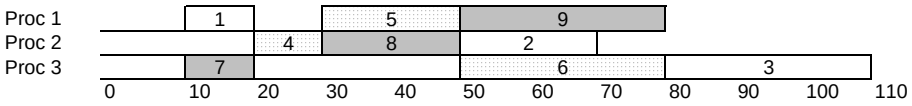

Figure 2-5: Gantt chart for a job shop with processors' time windows.

2.3.6 Limited buffers and blocking constraints

The JS assumes that there are unlimited buffers before and after each processor (assumption P4). Therefore, a job can wait in a buffer for an arbitrary time for its next operation to be performed. This assumption is valid in practice as long as the buffers' capacity is sufficient. Nevertheless, this might not be the case in industrial settings. Buffers are often limited in size for two reasons: (i) they require initial investments (especially in automated systems) and (ii) bigger buffers tolerate more work-in-process, which is considered as "waste" and should be reduced as much as possible according to the lean production philosophy. Technological constraints in some environments might not even allow any intermediate buffer at all. This results in a so-called *blocking* constraint that forces a job to stay on its current processor after its completion and thus block the processor during the waiting time for the job's next processor to be free. Observe that when a limited buffer is filled up to its capacity with in-process jobs, it can no longer store any other job, and processors needing the buffer to store in-process jobs become blocked as in the no-buffer (blocking) case. The JS with blocking constraints can also be referred to as *Blocking Job Shop (BJS)*.

Blocking can be found in various settings. For example, robot cells introduced in Example 1.1 typically have no space for temporary storage. In warehouses (Example 1.2), discharging palettes from a carriage to an aisle or a cross-aisle is not allowed, otherwise the aisles would be quickly filled up with discharged pallets and become unusable. Medical safety regulations do not allow taking a patient out of an operating room if a recovery bed is not ready, so any delayed

admission to the recovery room blocks the operating room from other surgeries (Example 1.3). Obviously, a train staying in a section prohibits other trains from entering the section (Example 1.4).

According to Hall and Sriskandarajah's survey of machine scheduling problems with blocking and no-wait in process [HS96], research on the job shop scheduling with blocking constraints has been scarce. More research activities have been recorded only recently with applications focused on the railway's conflict resolution problem in Example 1.4.

Example 2.5.

Consider Example 2.1 with blocking constraints. In a feasible solution below, blocking times are drawn in narrow bars. Observe for instance that job 3 cannot continue with operation 9 on processor 1 at time 50, which is the time operation 8 was completed on processor 2, because processor 1 is blocked by operation 5. This operation blocks processor 1 until time 60, when processor 3 for its job successor (operation 6) becomes free.

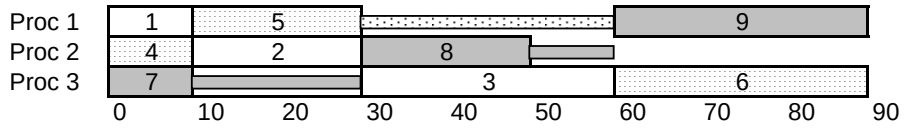

Figure 2-6: Gantt chart for a job shop with blocking constraints.

2.3.7 No-wait constraints

In several settings, a job once started should be processed continuously without any delay between two consecutive operations. This restriction, commonly referred to as *no-wait*, is mainly due to technological process requirements. The JS with no-wait constraints is termed *No-wait Job Shop (NWJS)*. No-wait constraints arise in many processing industries, including steel casting, concrete ware moulding, and chemical and pharmaceutical manufacturing, etc. For instance, in a galvanic plant, each job to be galvanized has to go through a series of tanks arranged in straight lines. Each line uses a horizontal hoist to move the jobs. The lines are interconnected by traversal hoists. If a job is done in a tank but its corresponding hoist is not available to take it to the next tank, then the job has to wait inside the tank. Any tank delay of more than one minute results in a serious quality degradation and hence a rejection of the job. Therefore, process regulations require that a job once started be processed continuously without any in-process delay [MMR99]. No-wait constraints can also be found in service environments. Lee et al. observed that many ambulatory centers perform outpatient cases in a back-to-back manner, which means that all patients are moved immediately from operating rooms to recovery rooms after their surgery without any admission delay [LdMH02]. Bianco et al. reported an air traffic control problem in which landing or take-off times of airplanes must be calculated

carefully to avoid traffic collision. Each airplane must follow strictly a predefined trajectory, which is divided into several air segments at fixed attitudes. An airplane can only enter an air segment if this segment is free. If a jet during its landing (take-off) finds out that the segment it is about to enter is still occupied, it cannot wait in between two segments to avoid a collision, which implies a no-wait constraint [BDG06].

Example 2.6.

The no-wait constraint is added to Example 2.1. A feasible schedule is given below. Observe from the resulting Gantt chart that even though processors 1 and 3 are idle in their first 10 time units, it is impossible to start operation 7 10 minutes or operation 1 30 minutes earlier due to the no-wait constraint.

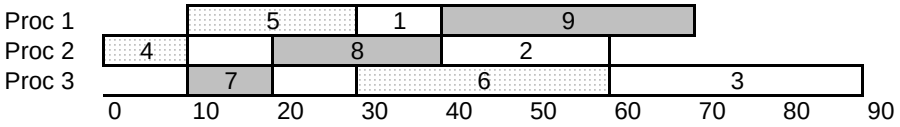

Figure 2-7: Gantt chart for a job shop with the no-wait constraint.

2.3.8 Generalized no-wait constraints

Under assumption P6, any operation can only start after the completion of its job predecessor (if any). Nevertheless, operations of a job can overlap. The requirement to have a fixed pattern of operations, overlapping or not, is termed *generalized no-wait* constraint. Generalized no-wait constraint is a direct result of the use of lot streaming in industries. Lot streaming splits a lot of identical parts into several sublots. All of these sublots must be processed continuously without any delay between two consecutive sublots. Once a sublot is completed, it must be transferred immediately to the next processing phase while the subsequent sublot of the same job is in process. In other words, the transfer lot's size is smaller than or equal to the processing lot's size. This contrasts to the standard job shop processing where the two sizes are the same. We can see that the no-wait constraint previously described is just a special case of the generalized no-wait constraint when the number of sublots of any job is one. Besides lot streaming, generalized no-wait may also arise due to technical requirements, e.g. in the chemical industry with batch-processing.

Although the JS with generalized no-wait (labeled as *GNWJS*) has been addressed as early as 1977 by Gröflin [Grö77], scheduling research in this stream has been scarce. The most recent paper on the subject by Alvarez-Valdes et al. discusses a real-life scheduling problem in a glass factory, where each job consists of four operations: furnace heating, blowing, cooling and packing in this order. The operations are tightly coupled: the first unit processed goes directly to the second operation right after the first operation, i.e. the start-start time lag between the

first and the second operation is equal to the processing time of one part in the first operation. The start-start time lag between the blowing and the cooling operations is calculated similarly, while the time to start the packing operation is determined by the length and the speed of the conveyor belt [AVFT+05].

Example 2.7.

Suppose lot streaming decisions have been made to split each job in Example 2.1 into two equal sublots. This leads to a fixed pattern of operations overlapping for each job as shown in Table 2-3. For example, processing the whole job 1 by processor 1 takes ten time units, but since lot streaming allows half of the completed parts of job 1 to transfer to processor 2 immediately after they are done, operation 2 can start five time units after operation 1 has started.

	J_1	J_2	J_3
Starting time of job	t_1	t_2	t_3
Starting time of operation			
First operation	$t_1 + 0$	$t_2 + 0$	$t_3 + 0$
Second operation	$t_1 + 5$	$t_2 + 5$	$t_3 + 5$
Third operation	$t_1 + 15$	$t_2 + 15$	$t_3 + 15$

Table 2-3: Operations overlapping pattern of Example 2.7.

Observe in the corresponding Gantt chart (the upper chart in Figure 2-8) that because of the generalized no-wait constraint, shifting an operation will move the whole job to which it belongs. In the lower Gantt chart in Figure 2-8, switching operation 9 after operation 1 moved the whole job 3.

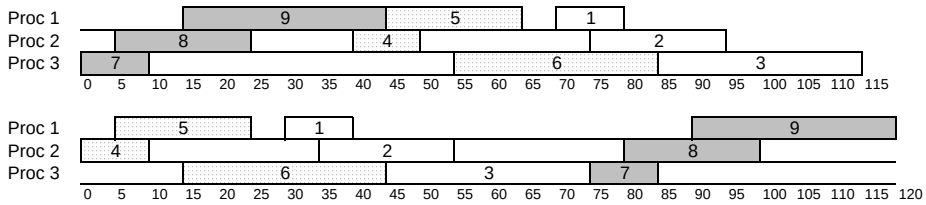

Figure 2-8: Gantt charts for a job shop with generalized no-wait constraints.

2.3.9 Processor flexibility

Assumption J3 states that each operation is to be done by a dedicated processor. This assumption removes resource allocation decisions from the scheduling problem, reducing it to a job sequencing problem. Nevertheless, it is rather common in practice to have alternative processors to perform an operation, although one of the processors might be preferred to the others for the operation. *Processor flexibility* represents the possibility to select for an operation its processor among several alternative processors. It is achieved through the processor's capability of performing different operation types and/or the availability of identical processors of the

same type. In manufacturing environments, processor flexibility is best observed in Flexible Manufacturing Systems (FMSs). An FMS typically contains several CNC machines along with an automatic material handling system. A robot cell in Example 1.1 can be a part of such an FMS. The machines are multipurpose in the sense that they can process different parts when they are equipped with the right tools supplied from tool magazines. In service environments, processor flexibility is often associated with labor flexibility, which is achieved by cross-training so that workers can obtain a broad range of skills for different tasks. For example, a general surgeon can handle a range of different outpatient surgery types (Example 1.3).

Example 2.8.

Consider the job shop instance in Example 2.1 with the following processor flexibility:

Proccessor/Operation	1	2	3	4	5	6	7	8	9
1	10				20				30
2	10	20	30	10				20	
3		20	30			30	10		

Table 2-4: Processing times in Example 2.8.

In the solution below, operation 1 is assigned to processor 1, operation 2 to processor 3, and operation 3 to processor 2. The processor for each of the other operations is already fixed. The resulting makespan is 60, which is 67% of the makespan obtained when there is no processor flexibility.

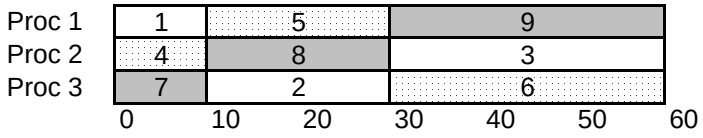

Figure 2-9: Gantt chart for a job shop with processor flexibility.

It is necessary to differentiate between two categories of processor flexibility: (i) processor flexibility with processor independency and (ii) processor flexibility with processor dependency. In the first case, the processor choice for an operation of a job is not dependent on the processor choices for other operations of the same job. As a result, a job J of m operations $i \in J$ has $\prod_{i \in J} |M_i|$ different routings, where M_i is the set of alternative processors for $i \in J$. The second case is more restrictive because a processor choice for an operation may narrow the range of processor choices for other operations of the same job. For instance, a critical patient having been operated in an operating room (OR) in a suite is moved to an Intensity Care Unit (ICU) in the same suite, not to another ICU located several blocks far away from the OR. The processor flexibility with processor dependency is associated with *alternative process plans* to process a job; each of the plans has its own routing. In manufacturing systems, process plans for a job are generated by Computer Aided Planning (CAP). To illustrate this flexibility category, consider a small instance of a job J with two operations J_1 and J_2 to be processed by four

processors $M_1, \ldots, M_4$. J_1 can be processed by M_1 or M_2 while J_2 can be performed by M_3 or M_4. Because of proximity, if J_1 is on M_1 then J_2 must be on M_3; and if J_1 is on M_2 then J_2 is on M_4. Consequently, there are only two possible routings for the job: $J_1(M_1) - J_2(M_3)$ and $J_1(M_2) - J_2(M_4)$, compared to four possible combinations in case of processor independency.

In scheduling literature, the JS with processor flexibility in the first flexibility category can be referred to as *Flexible Job Shop* (hereafter labeled as FJS) [MG00] or *JS with multipurpose machines* [HJT94]. The terms for the second processor flexibility category are *JS with alternative process plans* or *JS with alternative routings* [KE99].

2.3.10 Objective functions

Basically, an objective function is a function of the jobs' finishing times $C_J, J \in \mathcal{J}$, where C_J is defined as the time at which the processing of job J is completed and J leaves the system. Most often, an objective function is of *bottleneck* type $f_{\max}(C) := \max_{J\in\mathcal{J}}\{f_J(C_J)\}$ or *sum* type $\sum f(C) := \sum_{J\in\mathcal{J}} f_J(C_J)$. The objective function to minimize the makespan is defined as $C_{\max} = \max_{J\in\mathcal{J}}\{C_J\}$, thus it belongs to the bottleneck type. The objective function to minimize the total flow time $\sum_{J\in\mathcal{J}} C_J$ or the objective function to minimize the total weighted flow time $\sum_{J\in\mathcal{J}} w_J C_J$ are of the sum type. Different objective functions have different practical implications. Minimizing the makespan implies high utilization of processors because a given number of jobs is to be completed in the shortest time possible. Given a known fixed processing time for each operation, minimizing the total flow time implies minimizing the total queuing times, which might account for up to 80% of the total flow times, and hence reduces work-in-process and working capital tied up on the shop floor.

Assumption J5 states that there is no due date for any job. It is hardly true in today's time-based business, where meeting due dates is one of the management's top priorities to satisfy their customers. Objective functions concerning due dates can be expressed with the following functions:

1. lateness $L_J := C_J - d_J$
2. earliness $E_J := d_J - C_J$
3. tardiness $T_J := \max\{0, C_J - d_J\}$
4. unit penalty $U_J := \begin{cases} 0 \text{ if } C_J \leq d_J \\ 1 \text{ otherwise} \end{cases}$

where d_J is the due date for job $J \in \mathcal{J}$. Due-date based objective functions can be of the bottleneck type, e.g. minimizing maximum tardiness, or the sum type, e.g. minimizing total tardiness.

An objective function is said to be *regular* if it does not decrease with respect to increases in finishing times and *non-regular* otherwise. As an example, objective functions involving earliness are non-regular.

Of the above single-criterion objective functions, minimizing the makespan is the most common objective in scheduling research. Its popularity could be explained for three reasons. First, yielding high utilization is still one of the major management concerns, especially when processors are costly to acquire and operate such as operating rooms. Second, computing the makespan is facilitated by established graph algorithms to calculate the longest path's length in an associated solution graph. Third, research on the makespan job shop problem benefits from a large number of accumulated works on the subject over the past 50 years. It is expected that the dominance of makespan minimization will continue in the near future. Nevertheless, scheduling research on other objective functions, most notably on tardiness and multi-criteria objective functions, will probably receive higher attention in the future.

2.4 Evaluation of the JS' MILP formulations

The popularity of the mathematical programming approach to tackle scheduling problems could be attributed to several factors. First, it is a proven approach to solve many planning and scheduling problems in the manufacturing sector [PW06], where job-shop-like environments are frequently found. Second, the process of formulating a combinatorial optimization problem such as the JS can give insights that are useful for solution methodology development. Third, algorithmic advances and more powerful computing resources at cheaper cost have been improving commercial MILP solvers' capability, most notably over the past decade [Bix04]. Last, a large amount of detailed data necessary for decision making in scheduling is now available thanks to information technology development.

There are several alternative formulations for the JS and their performance may vary significantly. In this section, we try to answer the question "Which is the best formulation for the JS?" Previous studies on this issue, e.g. Liao and You (1992) [LY92] and Pan (1997) [Pan97], do not answer this question conclusively because of either faulty arguments or inadequate supporting computational experiments. We also address this question because the gained insight should help in developing formulations for extensions of the JS problem.

Mathematical programming formulations for the JS can be divided into two main groups depending on how the time horizon is handled:

1. Discrete-time formulations where the time is discretized in time periods (of usually equal length),

2. Continuous-time formulations where time is treated as continuous.

In this chapter, we study only continuous-time formulations. The main reason for this is that the number of time interval assignment variables in a discrete-time formulation (being either the Bowman or the Morten formulation [Pan97]) is determined by the value of operations' processing times. Consequently, in a JS instance with few jobs and processors, a long processing time for some operation, and relatively small processing durations for other operations may result in a very large-sized formulation [Pan97]. Therefore, unless the processing times of operations are highly homogeneous (e.g. all operations have the same processing time), discrete-time formulations have intrinsic limitations.

Continuous-time formulations can be further divided into two groups, which differ mainly in the way the operation sequence on each processor is expressed. Formulations in the first group, including the Manne formulation and its two variants namely the Liao-You and the adaptive Manne formulations, use disjunctive constraints to indicate the precedence relation between any two operations performed by the same processor, while the second group's formulations, including the Wagner formulation and its Wilson variant, assign each operation to a unique position in its processor's operation sequence.

The notations used in formulations for the JS are summarized in Table 2-5.

Notations		
Sets		*Where*
$\mathcal{J}$	Set of jobs J	
I	Set of operations	
M	Set of processors	
M_i	Set of processors assignable to $i \in I$	
A_J	Set of pairs of consecutive operations $(J_r, J_{r+1}), 1 \leq r < \lvert J\rvert$ of job $J \in \mathcal{J}$	
B	Set of pairs of operations $i, j \in I, i \neq j$ performed on a same processor $k, k \in M$	
I_k	Set of operations performed on processor $k \in M$	
Parameters		
p_i	Processing time of operation $i \in I_k$, $k \in M$	
H	Very large positive number	
H_{ij}	Very large positive number with respect to operations $i, j \in I_k$, $k \in M$	AM
Decision variables		
$y_{ij} =$	$\begin{cases} 1 \text{ if } i \text{ is sequenced before } j \text{ on some processor } k \in M_i \cap M_j, (i,j) \in B \\ 0 \text{ otherwise} \end{cases}$	MA, LY, AM
x_i	Starting time of operation $i \in I$	MA, LY, AM
x_τ	Starting time of dummy operation τ	
q_{ij}	Slack variables	LY
$\gamma^k_{iw} =$	$\begin{cases} 1 \text{ if operation } i \text{ is assigned to position } w \text{ on processor } k, 1 \leq w \leq \lvert I_k\rvert \\ 0 \text{ otherwise} \end{cases}$	WA, WI
t^k_w	Starting time of an operation occupying position w on k, $k \in M, 1 \leq w \leq \lvert I_k\rvert$	WA, WI
s^k_w	Idle time on processor k between the completion of an operation in sequence position w and the start of an operation in position $w+1, k \in M, 1 \leq w \leq \lvert I_k\rvert$	WI
s^k_0	Idle time on processor k before the start of the first operation in the sequence on $k \in M$	WI

MA=Manne, LY=Li-You, AM=Adaptive Manne, WA=Wagner, WI=Wilson
Used in all formulations otherwise

Table 2-5: Notations for the JS formulations

2.4.1 Manne formulation

The Manne formulation [Man60] defines sequence variable y_{ij} for any two operations performed on the same processor $i, j \in I$, $(i,j) \in B$ as:

$$y_{ij} = \begin{cases} 1 \text{ if } i \text{ precedes } j \text{ (not necessarily immediately)} \\ 0 \text{ otherwise} \end{cases}.$$

$$\text{Minimize } x_\tau \text{ subject to :} \tag{2-1}$$

$$x_j - x_i \geq p_i \text{ for all } (i,j) \in A_J, J \in \mathcal{J} \tag{2-2}$$

$$x_j + H(1 - y_{ij}) - x_i \geq p_i \text{ for all } (i,j) \in B \tag{2-3}$$

$$x_i + H y_{ij} - x_j \geq p_j \text{ for all } (i,j) \in B \tag{2-4}$$

$$x_\tau - x_i - p_i \geq 0 \text{ for all } i \in I \tag{2-5}$$

$$y_{ij} \in \{0,1\} \text{ for all } (i,j) \in B \tag{2-6}$$

$$x_i \geq 0 \text{ for all } i \in I \tag{2-7}$$

$$x_\tau \geq 0 \tag{2-8}$$

Objective function (2-1) minimizes the makespan, which is equal to the starting time of dummy operation τ of processing time zero. Constraints (2-2) ensure that any operation can start only after its job predecessor has been completed. Two capacity constraints (2-3) and (2-4) express the disjunctive constraints $x_j \geq x_i + p_i$ or $x_i \geq x_j + p_i$ for all $(i,j) \in B$. If i precedes j on processor k then $y_{ij} = 1$ and (2-3) becomes $x_j - x_i \geq p_i$ while (2-4) becomes redundant because of a large positive value H, commonly referred to as the *big-M*. On the other hand, if j precedes i then $y_{ij} = 0$ and (2-4) becomes $x_i - x_i \geq p_j$ while (2-3) becomes redundant. The value of H must be large enough to satisfy $H \geq x_i + p_i - x_j$ for all $i, j \in I_k$, for all $k \in M$. For this requirement, $H = \sum_{i \in I} p_i$ is sufficient. Constraints (2-5) force dummy operation τ to start after all other operations. The last three constraints define the domain for each variable.

2.4.2 Liao-You formulation

Liao and You modified the Manne formulation by introducing new slack variables q_{ij} into constraints (2-4) to obtain

$$H y_{ij} + x_i - x_j - p_j = q_{ij} \text{ for all } (i,j) \in B \tag{2-9}$$

Since $x_j - x_i = H y_{ij} - p_j - q_{ij}$, constraints (2-3) are now

$$q_{ij} \leq H - p_i - p_j \text{ for all } (i,j) \in B, i < j \tag{2-10}$$

and constraints (2-4) become

$$q_{ij} \geq 0 \text{ for all } (i,j) \in B, i < j \tag{2-11}$$

The Liao-You formulation [LY92] then comprises objective function (2-1) and constraints (2-2), (2-9), (2-10), (2-11), and (2-5) to (2-8). Liao and You's supporting argument is that even though their formulation has more continuous variables than the Manne formulation does, it has fewer *functional* (non-upper bound) constraints. Upper bound constraints for q_{ij} (2-10) can be handled efficiently by the simplex algorithm.

2.4.3 Adaptive Manne formulation

Both Manne and Liao-You formulations use a large positive value H for their capacity constraints. Observe that when an LP-relaxation-based commercial solver solves the Manne MILP formulation, a tighter value of H is more likely to force variable y_{ij} to take value 0 or 1 in the capacity constraints. We propose here a new scheme to tighten H: H is replaced by H_{ij} in constraints (2-3) and by H_{ji} in constraints (2-4) respectively. The resulting formulation, called *adaptive Manne formulation,* consists of objective function (2-1), constraints (2-2), (2-5) to (2-8), and the following two constraints

$$x_j + H_{ij}(1 - y_{ij}) - x_i \geq p_i \text{ for all } (i,j) \in B, i < j \tag{2-12}$$

$$x_i + H_{ji} y_{ij} - x_j \geq p_j \text{ for all } (i,j) \in B, i < j \tag{2-13}$$

The values of H_{ij} and H_{ji} are calculated as follows. In order to make constraints (2-13) redundant when $y_{ij} = 1$ (i precedes j), H_{ji} must satisfy $H_{ji} \geq x_j - x_i + p_j$ for all $(i,j) \in B, i < j$. Obviously, the value $H_{ji} = \max_{(i,j)\in B}(x_j - x_i) + \max_{j \in I} p_j$ satisfies this requirement. Similarly when j precedes i ($y_{ij} = 0$), a sufficient value of H_{ij} to keep constraints (2-12) redundant is $H_{ij} = \max_{(i,j)\in B}(x_i - x_j) + \max_{i \in I} p_i$. Observe that when i precedes j, $x_j - x_i \leq x_\tau - \mathit{optail}(j) - \mathit{ophead}(i)$ where $\mathit{optail}(j)$ is the sum of processing times of operation j and all of its job successors, while $\mathit{ophead}(i)$ is the sum of processing times of all job predecessors of operation i. For a given feasible schedule with makespan $\overline{C}$, e.g. a permutation schedule, we have $x_\tau \leq \overline{C}$. Thus, assuming $i = J_r^i$ and $j = J_s^j$, $H_{ij} = \max_{(i,j)\in B}(x_i - x_j) + \max_{i \in I} p_i \leq \overline{C} - \sum_{l=s}^{|J^j|} p_{J_l^j} - \sum_{l=1}^{r-1} p_{J_l^i} + \max_{i \in I} p_i$. Set $H_{ij} := \overline{C} - \sum_{l=s}^{|J^j|} p_{J_l^j} - \sum_{l=1}^{r-1} p_{J_l^i} + \max_{i \in I} p_i$. A value for H_{ij} is calculated similarly.

2.4.4 Wagner formulation

The original Wagner formulation addresses a somewhat more generalized scheduling problem than the JS in which each job goes through a series of processing stages and in each stage the job has one or several operations whose sequence is irrelevant [Wag59]. The Wagner formulation

views the scheduling problem as an assignment problem in which each operation has to be assigned to a unique position on its processor's operation sequence. Pan presents an adapted version of the Wagner formulation for the JS where each stage consists of only one processor and each job has only one operation in each of its stages [Pan97]. This adapted Wagner formulation has the following decision variables:

(1) $\gamma_{iw}^k = \begin{cases} 1 \text{ if operation } i \in I_k, k \in M, \text{is assigned to position } w, 1 \leq w \leq |I_k|, \\ 0 \text{ otherwise;} \end{cases}$

(2) t_w^k is the starting time of an operation occupying position w on processor k, $k \in M, 1 \leq w \leq |I_k|$;

(3) s_w^k, $k \in M, 1 \leq w \leq |I_k|$, is the idle time on processor k between the completion of the operation in the sequence position w and the start of the operation in position $w+1$;

(4) continuous variable s_0^k, $k \in M$, is the idle time on processor k before the start of the first operation in the sequence on k;

(5) the makespan is represented by the starting time x_τ of dummy operation τ.

The corresponding formulation is as follows.

$$\text{Minimize } x_\tau \text{ subject to :} \tag{2-14}$$

$$\sum_{w=1}^{|I_k|} \gamma_{iw}^k = 1 \quad \text{for all } i \in I_k, k \in M \tag{2-15}$$

$$\sum_{i \in I_k} \gamma_{iw}^k = 1 \quad \text{for all } k \in M, w = 1, \ldots, |I_k| \tag{2-16}$$

$$t_1^k = s_0^k \quad \text{for all } k \in M \tag{2-17}$$

$$t_w^k = \sum_{u=0}^{w-1} s_u^k + \sum_{u=1}^{w-1} \sum_{i \in I_k} p_i \gamma_{iu}^k \quad \text{for all } k \in M, w = 2, \ldots, |I_k| \tag{2-18}$$

$$t_w^k + p_i \leq t_{w'}^{k'} + H(1 - \gamma_{iw}^k) + H(1 - \gamma_{jw'}^{k'}) \tag{2-19}$$

$$\text{for all } i \in I_k, j \in I_{k'}, (i,j) \in A_J, J \in \mathcal{J}, k, k' \in M, w = 1, \ldots, |I_k|, w' = 1, \ldots, |I_{k'}|$$

$$x_\tau \geq t_{|I_k|}^k + \sum_{i \in I_k} p_i \gamma_{i|I_k|}^k \text{ for all } k \in M \tag{2-20}$$

$$\gamma_{iw}^k \in \{0,1\} \text{ for all } k \in M, i \in I_k, 1 \leq w \leq |I_k| \tag{2-21}$$

$$t_w^k \geq 0 \text{ for all } k \in M, 1 \leq w \leq |I_k| \tag{2-22}$$

$$s_w^k \geq 0 \text{ for all } k \in M, 1 \leq w \leq |I_k| \tag{2-23}$$

$$s_0^k \geq 0 \text{ for all } k \in M \tag{2-24}$$

$$x_\tau \geq 0 \tag{2-25}$$

Constraints (2-15) force each operation to occupy exactly one position on its processor's sequence. Constraints (2-16) make sure that each position on any processor is assigned to only

one operation. Constraints (2-17) and (2-18) are capacity constraints requiring that at any time there is at most one job processed on any processor. The processing order of any job is ensured by constraints (2-19). Note that a sufficient value of H to make (2-19) to be redundant when operation i and j are not assigned to position w on k and w' on k', respectively, is $H = \sum_{i \in I} p_i$. Constraints (2-20) enforce that dummy operation τ must start after the last operation on each processor. The last five constraints define the domain of variables.

2.4.5 Wilson formulation

Initially, Wilson modified the Wagner formulation for the flow shop scheduling problem. His formulation differs from the Wagner formulation in the way the precedence relation between any two consecutive operations performed on the same processor is formulated. Instead of using two equality constraints, the Wilson formulation uses a set of inequality constraints. The Wilson formulation for the JS, which results from a direct application of this modification to the Wagner job shop formulation above, has objective function (2-14), constraints (2-15), (2-16), (2-18) to (2-22), (2-25), and the following constraints:

$$t^k_{w+1} \geq t^k_w + \sum_{i \in I_k} p_i \gamma^k_{iw} \quad \text{for all } k \in M, w = 1, \ldots, |I_k| \tag{2-26}$$

Observe that because constraints (2-17) and (2-18) of the Wagner formulation are replaced by constraints (2-26), the explicit idle time variables used in constraints (2-17) and (2-18) are no longer needed in the Wilson formulation.

2.4.6 Evaluation of the JS' MILP formulations by a general-purpose solver

Evaluation of the five presented MILP formulations for the JS is conducted by analyzing their size complexity and comparing their performance with a general-purpose MILP solver on a set of benchmark instances. Size complexity of a formulation is expressed here by three metrics: (1) the number of binary variables, (2) the number of constraints, and (3) the number of continuous variables. The number of binary variables influences how many nodes are developed in the Branch-and-Bound algorithm. Thus increasing the number of binary variables could prolong the solution time. The number of constraints is related to the size of a base and influences the computing times of LP relaxation subproblems. The number of continuous variables also exercises some influence on the computing time of LP relaxation subproblems. For these reasons, it seems appropriate to characterize size by more than one metric, for example the number of binary variables as was in [Pan97]. Without loss of generality, assume that each job is processed by all processors, once by each of the processors. Table 2-6 summarizes the size complexities of the five JS formulations, where n is the number of jobs, m the number of processors, and mn is the number of operations.

Formulation	Binary variables	Continuous variables	
MA	$mn(n-1)/2$	$mn+1$	
LY	$mn(n-1)/2$	$mn(n+1)/2+1$	
AM	$mn(n-1)/2$	$mn+1$	
WA	mn^2	$2mn+1$	
WI	mn^2	$mn+1$	
Formulation	Total constraints	Functional constraints	Bound constraints
MA	$3mn^2/2+3mn/2-n+1$	mn^2-n	$mn(n+3)/2+1$
LY	$3mn^2-n+1$	$mn^2/2+mn/2-n$	$5mn^2/2-mn/2+1$
AM	$3mn^2/2+3mn/2-n+1$	mn^2-n	$mn(n+3)/2+1$
WA	$n^3(m-1)+mn^2+5mn+2m+1$	$n^3(m-1)+3mn+m$	$mn^2+2mn+m+1$
WI	$n^3(m-1)+mn^2+4mn+2m$	$n^3(m-1)+3mn+m$	mn^2+mn
MA=Manne, LY=Li-You, AM=Adaptive Manne, WA=Wagner, WI=Wilson			

Table 2-6: Size complexity of the JS' MILP formulations.

Several observations are drawn from this table:

1. The Manne family's formulations (MA, LY, and AM) have fewer binary variables and constraints, especially functional ones, than the Wagner family's members (WA and WI) do.

2. The number of continuous variables is almost identical between the two families, except in the Liao-You formulation where slack variables q_{ij} are used.

3. The size complexities of the Manne formulation and the Adaptive Manne formulation are identical because the latter tightens the former's capacity constraints only by modifying the value of H.

4. The Liao-You formulation reduces the number of functional constraints in the Manne formulation at the cost of additional real variables q_{ij} and bound constraints for q_{ij}.

5. The Wilson formulation improves size complexity of the Wagner formulation in the number of real variables and the number of bound constraints.

As the Wagner family's formulations have higher size complexity in comparison to the Manne family's formulations, we can expect the latter to outperform the former. Two formulations in the Wagner family would perform similarly. Among these two formulations, the Wilson formulation might perform slightly better than the Wagner one. It is difficult to compare the performance of the Manne-based formulations since they all have the same number of binary variables, while the impact of tightening the capacity constraints and introducing slack variables is difficult to assess unless a comprehensive computational experiment is made. Liao and You claimed that their formulation outperformed the Manne formulation. However, their supporting computational study was done on rather homogeneous instances of small sizes, which have not been used elsewhere as standard benchmark instances. Their claim would be more valid if it were confirmed by a more comprehensive computational study on so-called "standard"

JS benchmark instances. There are several test sets that are often used to evaluate solution methods developed for the JS, e.g. a set of three instances due to Fisher and Thompson that includes the famous 10×10 FT10 instance [AC91], a set of 40 instances due to Lawrence [Law84], a set of ten 10×10 instances by Applegate and Cook [AC91], a set of 80 instances by Taillard [Tai93], and some other sets [JM99]. We selected the Lawrence's instance set to reevaluate all MILP formulations for the JS because (1) it covers a rather wide range of problem sizes, (2) the number of instances (40) is not too big, and (3) many of its instances have been solved optimally. The set's instances are labeled $la01 - la40$ and grouped in eight equal subsets according to their sizes. Among these 40 instances, we limited our selection to only settled Lawrence instances with confirmed optimal makespans. Furthermore, we purposefully selected "easy" instances $la01 - la20$ and $la31 - la3$, for which the Branch-and-Bound (BnB) algorithm developed by Brucker, Jurish, and Sievers can obtain optimal makespans in less than 24 seconds with two exceptional computing times of 340 and 343 seconds for $la19$ and $la20$ respectively [BJS94]. We coded the JS formulations in the LPL mathematical modeling language version 4.99j [Hür07] and solved them by by general-purpose MILP solver CPLEX version 9.0 with default parameters [ILO]. The time limit was set at ten minutes, which is far beyond the times required to obtain optimality for these instances by the Brucker's BnB algorithm. Table 2-7 presents upper bounds and lower bounds obtained for each formulation and each instance, as well as the optimal makespan and the optimality time for each instance as reported in [BJS94]. For each formulation, we calculated four metrics: (1) its average relative error where *relative error =(Makespan found-Optimal makespan)/Optimal makespan× 100%,* (2) its average gap where *Gap=(Optimal makespan-Lower bound found)/Optimal makespan× 100%*, (3) its number of optimal makespans found without being proven optimal, and (4) its number of optimal solutions found. We then compared the JS formulations according to these metrics. Table 2-8 summarizes these metrics for all formulations.

As expected, all Manne-based formulations significantly outperformed the Wagner-based formulations. None of the Wagner family's members could yield a feasible solution for any instance within the allotted time while all Manne-based formulations did. As shown in Table 2-8, the Liao-You formulation under-performed the Manne and adaptive Manne formulations in all metrics. When being benchmarked with a more comprehensive test set, the Liao-You formulation did not perform as well as their authors claimed. The Manne formulation ranked first in three out of the four metrics, followed closely by the adaptive Manne formulation; but this order is reversed as for the number of the best makespans obtained. This suggests that simple tightening disjunctive constraints of the Manne formulation does not lead to the expected performance improvement for the Manne formulation. Note that more complicated tightening schemes that properly update the value of adaptive $big - M$ have not been attempted. Observe that the solver found optimal makespans for these two formulations in many instances without being able to prove optimality because of the low quality of the achieved lower bounds, which resulted in a gap of more than 33%. On the other hand, lower bounds by the Wagner and Wilson formulations,

Inst.	Size	BnB			Manne			Liao-You				Adaptive Manne				Wagner				Wilson			
	nxm	Opt	Time	UB	LB	RE	Gap	LB	RE	Gap	UB	LB	RE	Gap	UB	LB	RE	Gap	UB	LB	RE	Gap
la01	10x5	666	0	666	666.00	0.00	0.00	568.1	0.00	14.70	666	666	0.00	0.00	N/F	666	N/A	0.00	N/F	666	N/A	0.00
la02	10x5	655	3	655	655.00	0.00	0.00	566	0.00	13.59	655	655	0.00	0.00	N/F	635	N/A	3.05	N/F	635	N/A	3.05
la03	10x5	597	1	597	597.00	0.00	0.00	608	501.5	1.84	16.00	597	597	0.00	N/F	588	N/A	1.51	N/F	588	N/A	1.51
la04	10x5	590	4	590	590.00	0.00	0.00	590	556.1	0.00	5.75	590	590	0.00	N/F	537	N/A	8.98	N/F	537	N/A	8.98
la05	10x5	593	0	593	541.00	0.00	8.77	593	425	0.00	28.33	593	474	0.00	N/F	593	N/A	0.00	N/F	593	N/A	0.00
la06	15x5	926	0	926	521.35	0.00	43.70	926	517.1	0.00	44.15	926	531	0.00	N/F	926	N/A	0.00	N/F	926	N/A	0.00
la07	15x5	890	0	890	477.00	0.00	46.40	916	440	2.92	50.56	890	488	0.00	N/F	869	N/A	2.36	N/F	869	N/A	2.36
la08	15x5	863	0	864	498.00	0.12	42.29	863	474.3	0.00	45.04	863	490	0.00	N/F	863	N/A	0.00	N/F	863	N/A	0.00
la09	15x5	951	0	951	511.00	0.00	46.27	951	498	0.00	47.63	951	510	0.00	N/F	951	N/A	0.00	N/F	951	N/A	0.00
la10	15x5	958	0	958	544.00	0.00	43.22	958	529.2	0.00	44.76	958	539	0.00	N/F	958	N/A	0.00	N/F	958	N/A	0.00
la11	20x5	1222	0	1222	495.56	0.00	59.45	1279	499.6	4.66	59.11	1222	509	0.00	N/F	1222	N/A	0.00	N/F	1222	N/A	0.00
la12	20x5	1039	1	1039	470.00	0.00	54.76	1096	470	5.49	54.76	1039	471	0.00	N/F	1039	N/A	0.00	N/F	1039	N/A	0.00
la13	20x5	1150	0	1151	513.47	0.09	55.35	1219	485	6.00	57.83	1150	508.4	0.00	N/F	1150	N/A	0.00	N/F	N/F	N/A	N/A
la14	20x5	1292	0	1292	518.06	0.00	59.90	1292	521	0.00	59.68	1292	518.1	0.00	N/F	1292	N/A	0.00	N/F	1292	N/A	0.00
la15	20x5	1207	4	1270	482.00	5.22	60.07	1353	474	12.10	60.73	1251	497	3.65	58.82	N/F	N/A	0.00	N/F	1207	N/A	0.00
la16	10x10	945	58	946	845.00	0.11	10.58	968	783	2.43	17.14	945	908	0.00	3.92	N/F	N/A	30.16	N/F	660	N/A	30.16
la17	10x10	784	15	784	774.00	0.00	1.28	796	664.7	1.53	15.22	784	720	0.00	8.16	N/F	N/A	12.88	N/F	683	N/A	12.88
la18	10x10	848	64	848	848.00	0.00	0.00	862	740	1.65	12.74	853	777.4	0.59	8.33	N/F	N/A	26.53	N/F	623	N/A	26.53
la19	10x10	842	340	842	842.00	0.00	0.00	842	749	0.00	11.05	842	842	0.00	0.00	N/F	N/A	18.65	N/F	685	N/A	18.65
la20	10x10	902	343	902	875.00	0.00	2.99	907	886.7	0.55	1.70	903	871	0.11	3.44	N/F	N/A	17.52	N/F	744	N/A	17.52
la31	30x10	1784	8	1966	767.40	10.20	56.98	2280	770	27.80	56.84	2012	763.5	12.78	57.21	N/F	N/A	N/A	N/F	N/F	N/A	N/A
la32	30x10	1850	1	2199	828.38	18.86	55.22	2419	779	30.76	57.89	2134	820	15.35	55.68	N/F	N/A	N/A	N/F	N/F	N/A	N/A
la33	30x10	1719	77	1958	745.00	13.90	56.66	2296	745	33.57	56.66	2072	745	20.54	56.66	N/F	N/A	N/A	N/F	N/F	N/A	N/A
la34	30x10	1721	15	2036	689.00	18.30	59.97	2269	672	31.84	60.95	2109	689	22.55	59.97	N/F	N/A	N/A	N/F	N/F	N/A	N/A
la35	30x10	1888	24	2172	709.19	15.04	62.44	2399	674	27.07	64.30	2240	709.2	18.64	62.44	N/F	N/A	N/A	N/F	N/F	N/A	N/A

BnB = Branch-and-Bound method of Bruckers et al., Opt = optimal makespan, Time = optimality time in seconds, UB = upper bound (makespan obtained), LB = lower bound obtained by CPLEX
RE = relative error, N/F = not found, N/A = not applicable

Table 2-7: Performance of the JS formulations.

Formulations	Average relative error (%)	Average gap (%)	Nr. of best makespans found	Nr. of optimal makespans
Manne	3.27	33.05	16	6
Liao-You	7.61	38.28	10	0
Adaptive Manne	3.77	33.78	17	5

Table 2-8: Summary of performance of the JS formulations.

(if achieved) were better (at 6.08% and 6.40%, respectively). There was no evidence to say that the Wilson formulation performed better than the Wagner one as expected. Note that while performing poorly for the JS, the Wagner-based formulations outperformed the Manne-based formulations for the permutation flow shop problem as reported in [TSG04].

Due to its superiority in comparison to the other formulations, the Manne formulation was selected as the base to formulate generalized job shop scheduling problems in the following section. Hereinafter we shall refer to it as *Classical Job Shop (CJS) formulation*.

2.5 Formulations of practical job shop scheduling problems

This section first presents MILP formulations for 1-feature JS extension problems for the various complexifying features mentioned in Section 2.3. 1-feature formulations can then be used as "building blocks" to formulate practical job shop scheduling problem which possesses more than one of the studied features. This is illustrated by developing a formulation for a complex JS with two additional features.

2.5.1 Formulations of 1-feature JS extension problems

Sequence-dependent setup times

Extending the Manne formulation for the JS to cover the feature of sequence-dependent setup times is rather straight forward. Let $p^s_{ij} \geq 0$ be the sequence-dependent setup times between operation i and its succeeding operation j on the same processor $k = \mu(i) = \mu(j), i, j \in I, k \in M$. Let $p^s_{\sigma i}$ denote the setup time for operation $i \in I_k$ when i is the first operation in an operation sequence on processor k. Also denote by $p^s_{i\tau}$ the setup time for operation $i \in I_k$ when i is the last operation in the sequence on k. The MILP formulation for the job shop problem with sequence-dependent setup times, labeled as the $SDSJS$ formulation, is derived from the CJS formulation by including corresponding setup times into four constraints (2-3)-(2-6) while keeping the objective function and other constraints of the CJS. The full SDJS formulation is

given below.

$$\text{Minimize } x_\tau \text{ subject to :} \tag{2-27}$$

$$x_j - x_i \geq p_i \text{ for all } (i,j) \in A_J, J \in \mathcal{J} \tag{2-28}$$

$$x_j + H(1-y_{ij}) - x_i \geq p_i + p^s_{ij} \text{ for all } (i,j) \in B \tag{2-29}$$

$$x_i + Hy_{ij} - x_j \geq p_j + p^s_{ji} \text{ for all } (i,j) \in B \tag{2-30}$$

$$x_i \geq p^s_{\sigma i} \text{ for all } i \in I \tag{2-31}$$

$$x_\tau - x_i \geq p_i + p^s_{i\tau} \text{ for all } i \in I \tag{2-32}$$

$$y_{ij} \in \{0,1\} \text{ for all } (i,j) \in B, i < j \tag{2-33}$$

$$x_i \geq 0 \text{ for all } i \in I \tag{2-34}$$

$$x_\tau \geq 0 \tag{2-35}$$

Consider three operations u, v, and w, which are sequenced in this order on some processor k. Then $y_{uv} = y_{vw} = y_{uw} = 1$, and from (2-29) we have (i) $x_v - x_u \geq p_u + p^s_{uv}$, (ii) $x_w - x_v \geq p_v + p^s_{vw}$, and (iii) $x_w - x_u \geq p_u + p^s_{uw}$. From (i) and (ii), $x_w - x_u \geq p_v + p^s_{vw} + p_u + p^s_{uv}$ (iv). Since u and w are not performed consecutively, (iii) should be implied by (i) and (ii). This can be achieved by asking $p_v + p^s_{vw} + p_u + p^s_{uv} \geq p_u + p^s_{uw}$ or $p_v + (p^s_{uv} + p^s_{vw} - p^s_{vw}) \geq 0$ (v). When the *triangle* inequality ($p^s_{uv} + p^s_{vw} \geq p^s_{uw}$) is assumed for the SDSJS then (v) clearly holds. Further, this assumption allows us to remove transitive arcs in the associated disjunctive graph in calculating the length of the longest path from σ to τ (the critical path).

This SDJS formulation can be found in several papers (see [ANCK06] for references).

Multiple processors

Let M_i be the set of (one or several) processors to perform operation $i \in I$. Two operations $i, j \in I$ can be in a processor's capacity conflict if they share some common processor $k \in M_i \cap M_j$. We can extend the CJS formulation to obtain a Job Shop with Multiple Processors formulation ($MPJS$) that covers the multiprocessor feature with just a minor change by defining the conflict set B as $B = \{(i,j) : i,j \in I, M_i \cap M_j \neq \emptyset\}$.

Job release times

Formulating job release times is done by identifying a set I^{first} of the jobs' first operations, i.e. $I^{first} = \{i \in I : i = J_1, J \in \mathcal{J}\}$, and adding the following constraints to the CJS

$$x_i \geq r_i \text{ for all } i \in I^{first}, \tag{2-36}$$

where r_i is the release time of job J^i. The resulting formulation is labeled $JRJS$.

Processor time windows

There are two approaches to formulate the feature of processor time windows. The first approach fills each period during which a processor is unavailable by a dummy single-operation job. The processing time of a dummy operation equals the length of the corresponding unavailable period and its starting time is the beginning of the period. The dummy operations are included in a set of dummy operations I^{dum}, which is then added to the existing set of operations I. We include the following constraints to the CJS:

$$x_i = u_i \quad \text{for all } i \in I^{dum}, \tag{2-37}$$

where u_i is the start of the unavailability period associated with dummy operation $i \in I^{dum}$.

The second formulation approach can be applied when there is a single period of availability $[s_k, e_k]$ for each processor $k \in M$. We add the following constraints to the CJS:

$$x_i \geq s_k \quad \text{for all } i \in I_k, k \in M \tag{2-38}$$

$$x_i \leq e_k \quad \text{for all } i \in I_k, k \in M \tag{2-39}$$

We label the resulting formulation as $TWJS$.

Blocking constraints

Blocking constraints enforce that if a job is completed on a processor and its next processor is still busy, then the job stays on the the current processor and blocks it from processing another job until its next processor becomes available. The two processor capacity constraints of the CJS formulation, constraints (2-3) and (2-4), do not enforce that the next operation on the processor starts only after the commencement of the job successor of the current operation. Therefore, the CJS formulation is not straightforwardly extensible to cover blocking constraints. One way to achieve this is to introduce, besides starting time variables, also variables expressing each operation's finishing time, i.e. the time the operation's job leaves its current processor. This finishing time may come after the time the job processing on the processor is completed due to possible blocking. Denote by x_i^t and x_i^h the starting and finishing times of operation $i \in I$, respectively. We retain the sequence decision variables y_{ij} and other parameters of the CJS and present below the formulation for the job shop with blocking constraints or *Blocking Job Shop*

(BJS).

$$\text{Minimize } x_\tau \text{ subject to :} \tag{2-40}$$
$$x_i^h - x_i^t \geq p_i \text{ for all } i \in I \tag{2-41}$$
$$x_j^t - x_i^h = 0 \quad \text{for all } (i,j) \in A_J, J \in \mathcal{J} \tag{2-42}$$
$$x_j^t + H(1 - y_{ij}) - x_i^h \geq 0 \text{ for all } i,j \in I_k, i < j, k \in M \tag{2-43}$$
$$x_i^t + H y_{ij} - x_j^h \geq 0 \text{ for all } i,j \in I_k, i < j, k \in M \tag{2-44}$$
$$x_\tau - x_i^h \geq p_i \quad \text{for all } i \in I \tag{2-45}$$
$$y_{ij} \in \{0,1\} \quad \text{for all } i,j \in I, i < j, M_i \cap M_j \neq \emptyset \tag{2-46}$$
$$x_i^t, x_i^h \geq 0 \quad \text{for all } i \in I \tag{2-47}$$
$$x_\tau \geq 0 \tag{2-48}$$

The objective function is to minimize the makespan. Constraints (2-41) ensure that a job can only leave its current processor after its operation by the current processor has been completed. Constraints (2-42) require that the next operation of a job start without delay once the job has finished its sojourn on the processor of the current operation. Constraints (2-43) and (2-44) are capacity constraints, which ensure that the processor is blocked until its current job leaves for the next processing. These two constraints are mutually redundant upon the binary values of y_{ij}. If $y_{ij} = 1$, i.e. i precedes j on k, then (2-43) becomes $x_j^t - x_i^h \geq 0$, which requires j to starts on k only after the job of i leaves k. The case where $y_{ij} = 0$ is interpreted analogously. The rest of the constraints define the decision variables' domain.

Limited buffers

The JS with a buffer of limited size b can be formulated as a BJS in either of the two following ways, depending on the buffer's type:

1. Sequential buffer where each job traverses through b one-job buffers in series (e.g. conveyor belts) in that order. In this case, assuming that the conveyor traversal time is negligible, any job entering the buffer can be seen as having b dummy zero operations in series by b dummy processors;

2. Parallel buffer where each job can be placed in any of the b available space of the buffer (e.g. shop floor). In this case, any job entering the buffer can be seen as having one dummy zero-time operation performed by one of b dummy parallel and identical processors. Scheduling with processor flexibility is to be studied later.

In both cases, the buffer's upstream processor is blocked when all buffer's slots are occupied. Figure 2-10 illustrates instances of these buffer types with $n = 3$ jobs, $m = 4$ processors, and

$b = 2$. Routings for job 1, 2, and 3 are $M_1 - M_4$, $M_3 - M_4$, and $M_2 - M_4$ respectively. Buffer b is considered as a composition of two sub-buffers b_1 and b_2, both are of capacity one.

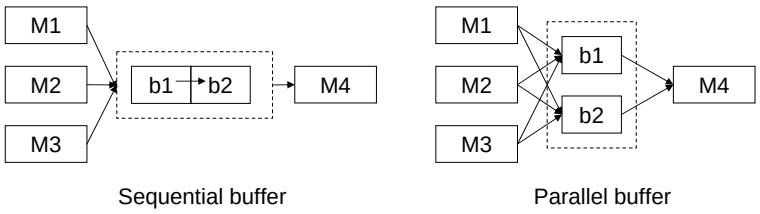

Figure 2-10: Modeling the JS with limited buffers as the BJS.

No-wait constraints

We adapt the CJS formulation for the JS with no-wait constraints ($NWJS$) by changing the inequality constraints (2-2) to the following equality constraints

$$x_j - x_i = p_i \text{ for all } (i,j) \in A_J, J \in \mathcal{J} \tag{2-49}$$

and keeping the rest of the CJS formulation. The resulting formulation comprises objective function (2-1), constraints (2-49), and constraints (2-3) to (2-8).

Generalized no-wait constraints

Let o_i be the time lag between the starting time of operation i and the starting time of the first operation of the operation's job (denoted as J^i) and s_J be the decision variable to express the starting time of job J. As the starting time of a job defines the starting time of each of the job's operations, a formulation for the JS with generalized no-wait constraints ($GNWJS$) is obtained by modifying the precedence constraints (2-2):

$$x_i - s_{J^i} = o_i \text{ for all } i \in J^i \tag{2-50}$$

Job transfer times

Job transfer times can be easily integrated into the BJS, the NWJS or the GNWJS formulations. For example, the BJS with job transfer times is formulated as follows. Let p_i^t and p_i^h be respectively the hand-over and take-over times associated with operation i. To incorporate job transfer times in the BJS formulation, add the hand-over and take-over times to constraints (2-41),(2-43), and (2-44) as follows:

$$x_i^h - x_i^t \geq p_i^t + p_i \text{ for all } i \in I \tag{2-51}$$

$$x_j^t + H(1 - y_{ij}) - x_i^h \geq p_i^h \text{ for all } i, j \in I_k, i < j, k \in M \tag{2-52}$$

$$x_i^t + Hy_{ij} - x_j^h \geq p_j^h \text{ for all } i, j \in I_k, i < j, k \in M \tag{2-53}$$

$$x_\tau - x_i^h \geq p_i + p_i^h \quad \text{for all } i \in I \tag{2-54}$$

The resulting formulation is termed *BJST*. Integrating job transfer times in the NWJS and the GNWJS formulations can be carried out similarly. The resulting formulations are not shown here.

Processor flexibility

Decisions on which processor is allocated to which operation are formulated with the help of additional variables:

$$z_{ik} = \begin{cases} 1 \text{ if operation } i \text{ is assigned to processor } k \in M_i \\ 0 \text{ otherwise.} \end{cases}$$

Let p_{ik} be the processing time of operation i by processor $k \in M_i$. The first mathematical formulation for the Flexible Job Shop (FJS), labeled *FJS-1*, is proposed in [GKZ93] as follows:

$$\text{Minimize } x_\tau \text{ subject to :} \tag{2-55}$$

$$x_j - x_i \geq \sum_{k \in M_i} p_{ik} z_{ik} \quad \text{for all } (i, j) \in A_J, J \in \mathcal{J} \tag{2-56}$$

$$x_j + H(1 - y_{ij}) - x_i + H(2 - z_{ik} - z_{jk}) \geq p_{ik} \tag{2-57}$$
$$\text{for all } i, j \in I, i < j, k \in M_i \cap M_j$$

$$x_i + Hy_{ij} - x_j + H(2 - z_{ik} - z_{jk}) \geq p_{jk} \tag{2-58}$$
$$\text{for all } i, j \in I, i < j, k \in M_i \cap M_j$$

$$x_\tau - x_i - \sum_{k \in M_i} p_{ik} z_{ik} \geq 0 \quad \text{for all } i \in I \tag{2-59}$$

$$\sum_{k \in M_i} z_{ik} = 1 \text{ for all } i \in I \tag{2-60}$$

$$z_{ik} \in \{0, 1\} \text{ for all } i \in I, k \in M_i \tag{2-61}$$

$$y_{ij} \in \{0, 1\} \quad \text{for all } (i, j) \in B, i < j \tag{2-62}$$

$$x_i \geq 0 \quad \text{for all } i \in I \tag{2-63}$$

$$x_\tau \geq 0 \tag{2-64}$$

If at least one of the two operations i and j, $M_i \cap M_j \neq \emptyset$, is not assigned to some processor $k \in M_i \cap M_j$, then $(2 - z_{ik} - z_{jk}) \geq 1$ and both constraints (2-57) and (2-58) are redundant

because of the high value of H. Only when both operations i and j are assigned to k do these two constraints become the disjunctive constraints as previously explained. In order to hold constraints (2-57) and (2-58) redundant when needed, H should satisfy $H \geq x_i + p_{ik} - x_j$ for all $i, j \in I, k \in M_i \cap M_j$. A value meeting this requirement is $H = \sum_{i \in I} \max_{k \in M_i}(p_{ik})$. Constraints (2-60) require that each operation is assigned to only one processor.

The big number H has a coefficient of three in the capacity constraints of the FJS-1 formulation. As H of large value is suspected to cause the CJS' LP relaxation to perform poorly, increasing the coefficient of H could probably worsen the issue. For this reason, a new formulation is proposed to keep the coefficient of H in the two capacity constraints at one as in the CJS. This formulation, labeled FJS-2, applies binary variables y_{ij} for all ordered pairs (i, j), with $M_i \cap M_j \neq \emptyset$.

The formulation is below.

$$\text{Minimize } x_\tau \text{ subject to :} \tag{2-65}$$

$$x_j - x_i \geq \sum_{k \in M_i} p_{ik} z_{ik} \quad \text{for all } (i,j) \in A_J, J \in \mathcal{J} \tag{2-66}$$

$$x_j + H(1 - y_{ij}) - x_i \geq p_{ik} z_{ik} \text{ for all } i, j \in I, i < j, k \in M_i \cap M_j \tag{2-67}$$

$$x_i + H(1 - y_{ji}) - x_j \geq p_{jk} z_{jk} \text{ for all } i, j \in I, i < j, k \in M_i \cap M_j \tag{2-68}$$

$$x_\tau - x_i - \sum_{k \in M_i} p_{ik} z_{ik} \geq 0 \quad \text{for all } i \in I \tag{2-69}$$

$$\sum_{k \in M_i} z_{ik} = 1 \text{ for all } i \in I \tag{2-70}$$

$$y_{ij} + y_{ji} \geq z_{ik} + z_{jk} - 1 \text{ for all } i, j \in I, k \in M_i \cap M_j \tag{2-71}$$

$$z_{ik} \in \{0, 1\} \text{ for all } i \in I, k \in M_i \tag{2-72}$$

$$y_{ij} \in \{0, 1\} \quad \text{for all } i, j \in I, M_i \cap M_j \neq \emptyset \tag{2-73}$$

$$x_i \geq 0 \quad \text{for all } i \in I \tag{2-74}$$

$$x_\tau \geq 0 \tag{2-75}$$

If $z_{ik} + z_{jk} \leq 1$ then constraints (2-67), (2-68), and (2-71) become redundant. If $z_{ik} = z_{jk} = 1$ then constraints (2-71) enforce that $y_{ij} + y_{ji} = 1$, i.e. exactly one of the two variables should take value one. Therefore, exactly one of (2-67) and (2-68) is active while the other one is redundant. Note that the coefficient reduction of H in the capacity constraints comes at the price of doubling the number of binary variables y_{ij} and increasing the number of constraints ((2-71)).

Objective functions

Formulating the JS problem with an objective function other than the makespan minimization is done by replacing the makespan minimization objective function (2-1) and makespan constraints (2-6) of the CJS formulation by the objective function and constraints corresponding to the new objective function. For instance, the formulation with the objective function to minimize the maximum lateness $L_{\max} := \max_{J\in\mathcal{J}} L_J$ is given below:

$$\text{Minimize } L_{\max} \text{ subject to :} \tag{2-76}$$

$$x_{J_{|J|}} + p_{J_{|J|}} - d_J \leq L_{\max} \text{ for all } J \in \mathcal{J} \tag{2-77}$$

$$\text{Constraints (2-2)-(2-4), (2-6)-(2-8)}$$

If the objective function is to minimize the total weighted tardiness, then we have:

$$\text{Minimize } \sum_{J\in\mathcal{J}} w_J T_J \text{ subject to :} \tag{2-78}$$

$$x_{J_{|J|}} + p_{J_{|J|}} - d_J \leq T_J \text{ for all } J \in \mathcal{J} \tag{2-79}$$

$$T_J \geq 0 \text{ for all } J \in \mathcal{J} \tag{2-80}$$

$$\text{Constraints (2-2)-(2-4),(2-6)-(2-8)}$$

Constraints (2-79) and (2-80) ensure that $T_J = \max\{0, C_J - d_J\}$, where $C_J = x_{J_{|J|}} + p_{J_{|J|}}$ is the finishing time of job J and d_J is its due date.

2.5.2 Formulation of complex job shop scheduling problems

Formulation of the Multimode Job Shop Scheduling problem

Job shop problems in practice often have more than one of the practical features presented in Section 2.3. An example of such a complex job shop scheduling problems is the *Multimode Job Shop Scheduling problem (MMJS)*, which was introduced by Brucker and Neyer (1998) [BN98]. The problem is stated as follows. Let $\mathcal{J}, I$, and M be the set of jobs, operations, and processors respectively. The processing of an operation i requires several processors simultaneously; the set of these processors forms a *mode*. Denote by $\mathcal{A}$ the set of all q possible modes $\mathcal{A}_k, 1 \leq k \leq q$. Associated with each operation $i \in I$ is a set of assignable modes $\mathcal{A}^i \subseteq \mathcal{A}, \cup_{i\in I}\mathcal{A}^i = \mathcal{A}$. The processing time of operation i in mode $\mathcal{A}_k \in \mathcal{A}^i$ is p_{ik}. Denote by $\mu(i)$ the processing mode assigned to operation i. Finding a feasible schedule that minimizes the makespan consists of assigning a processing mode to each operation and sequencing jobs for each processor. It is obvious that the MMJS is the JS complexified by two additional features: multiprocessor (each processing mode might contain more than one processor) and processor flexibility (there may be more than one assignable mode for each operation). To put it differently, the MPJS is a

special case of the MMJS where each operation has only one assignable mode, and the FJS is also a special case of the MMJS where each mode in the MMJS contains only one processor.

Based on the MPJS and the FJS-1 formulations, we propose an MILP formulation for the MMJS as follows. The mode assignment decision variables z_{ik} is defined as:

$$z_{ik} = \begin{cases} 1 \text{ if operation } i \text{ is assigned to mode } \mathcal{A}_k \in \mathcal{A}^i, \text{i.e. } \mu(i) = \mathcal{A}_k, \\ 0 \text{ otherwise} \end{cases}$$

Two operations i and j are said to be in conflict if i is assigned to mode $\mathcal{A}_k$, j is assigned to mode $\mathcal{A}_l$ and $\mathcal{A}_k \cap \mathcal{A}_l \neq \emptyset$. An operation uses all processors in its assigned mode at the same time; hence if i precedes j on processor $h \in \mathcal{A}_k \cap \mathcal{A}_l$ then i precedes j on all processors in $\mathcal{A}_k \cap \mathcal{A}_l$. Let x_i and x_τ be the starting times of operation $i \in I$ and dummy end operation τ respectively. The sequence decision variable y_{ij} is now defined for all unordered pairs (i, j) such that $\exists \mathcal{A}_k \in \mathcal{A}^i, \exists \mathcal{A}_l \in \mathcal{A}^j$ and $\mathcal{A}_k \cap \mathcal{A}_l \neq \emptyset_k$ as

$$y_{ij} = \begin{cases} 1 \text{ if operation } i \text{ precedes operation } j \\ 0 \text{ otherwise} \end{cases}$$

The full MMJS formulation is presented below:

$$\text{Minimize } x_\tau \text{ subject to :} \tag{2-81}$$

$$x_j - x_i \geq \sum_{\mathcal{A}_k \in \mathcal{A}^i} p_{ik} z_{ik} \quad \text{for all } (i,j) \in A_J, J \in \mathcal{J} \tag{2-82}$$

$$x_j + H(1 - y_{ij}) - x_i + H(2 - z_{ik} - z_{jl}) \geq p_{ik} \tag{2-83}$$
$$\text{for all } i,j,k,l \text{ with } i,j \in I, \mathcal{A}_k \in \mathcal{A}^i, \mathcal{A}_l \in \mathcal{A}^j, \text{ and } \mathcal{A}_k \cap \mathcal{A}_l \neq \emptyset$$

$$x_i + H y_{ij} - x_j + H(2 - z_{ik} - z_{jl}) \geq p_{jl} \tag{2-84}$$
$$\text{for all } i,j,k,l \text{ with } i,j \in I, \mathcal{A}_k \in \mathcal{A}^i, \mathcal{A}_l \in \mathcal{A}^j, \text{ and } \mathcal{A}_k \cap \mathcal{A}_l \neq \emptyset$$

$$x_\tau - x_i - \sum_{\mathcal{A}_k \in \mathcal{A}^i} p_{ik} z_{ik} \geq 0 \quad \text{for all } i \in I \tag{2-85}$$

$$\sum_{\mathcal{A}_k \in \mathcal{A}^i} z_{ik} = 1 \text{ for all } i \in I \tag{2-86}$$

$$z_{ik} \in \{0,1\} \text{ for all } i \in I, \mathcal{A}_k \in \mathcal{A}^i \tag{2-87}$$

$$y_{ij} \in \{0,1\} \quad \text{for all } i,j \in I\text{: } i < j, \exists \mathcal{A}_k \in \mathcal{A}^i, \exists \mathcal{A}_l \in \mathcal{A}^j \text{ and } \mathcal{A}_k \cap \mathcal{A}_l \neq \emptyset_k \tag{2-88}$$

$$x_i \geq 0 \quad \text{for all } i \in I \tag{2-89}$$

$$x_\tau \geq 0 \tag{2-90}$$

The second H-term in capacity constraints (2-83) and (2-84) ensures that operations i and j have a resource conflict only if the two modes assigned to them share at least one processor. In this case, the two constraints express the alternative relations as previously explained. Constraints (2-86) make sure that each operation is assigned to one processing mode. Other constraints

have already been explained.

Example 2.9.

Below are three modes and mode-dependent processing times for the job shop instance in Example 2.1.

Processor/Mode	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$
1	1	1	0
2	1	0	1
3	0	1	1

Table 2-9: Processors in modes.

Mode/Operation	1	2	3	4	5	6	7	8	9
$\mathcal{A}_1$	10				20				30
$\mathcal{A}_2$	10	20	20	10				20	
$\mathcal{A}_3$		10	30			30	10		

Table 2-10: Mode-dependent processing times.

Observe for instance that operation 2 can be performed either in mode 2 that uses simultaneously processors 1 and 3 or in mode 3 that employs processors 2 and 3. A feasible solution of the instance is displayed in Figure 2-11.

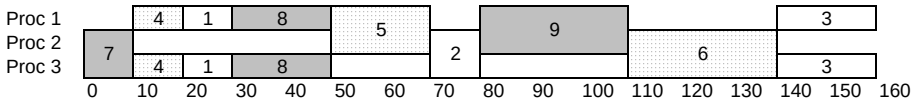

Figure 2-11: Gantt chart for Example 2.9.

A structure tree to analyze and formulate complex job shop scheduling problems

We have seen how the MMJS, a complex job shop scheduling problem, was analyzed and formulated based on two 1-feature formulations for the FJS and MPJS. In the graph below, a structure tree is presented to facilitate the process to analyze and formulate a complex scheduling problem. The tree displays several single complexifying features (in solid boxes) together with their associated MILP formulation (in dashed boxes) and some multi-feature problems (e.g. 2-feature and 3-feature problems) that are composed from 1-feature problems (see solid arcs) and formulated correspondingly (see dashed arcs). For example, a complex scheduling problem called Generalized Blocking Job Shop (*GBJS*) in [Kli01] is a 3-feature problem having sequence-dependent setup times, blocking constraints, and transfer times features. Note that the transfer times feature should be incorporated with one of the following features: blocking, no-wait, or generalized no-wait constraints. The tree can be extended by adding more features and/or combining more single features.

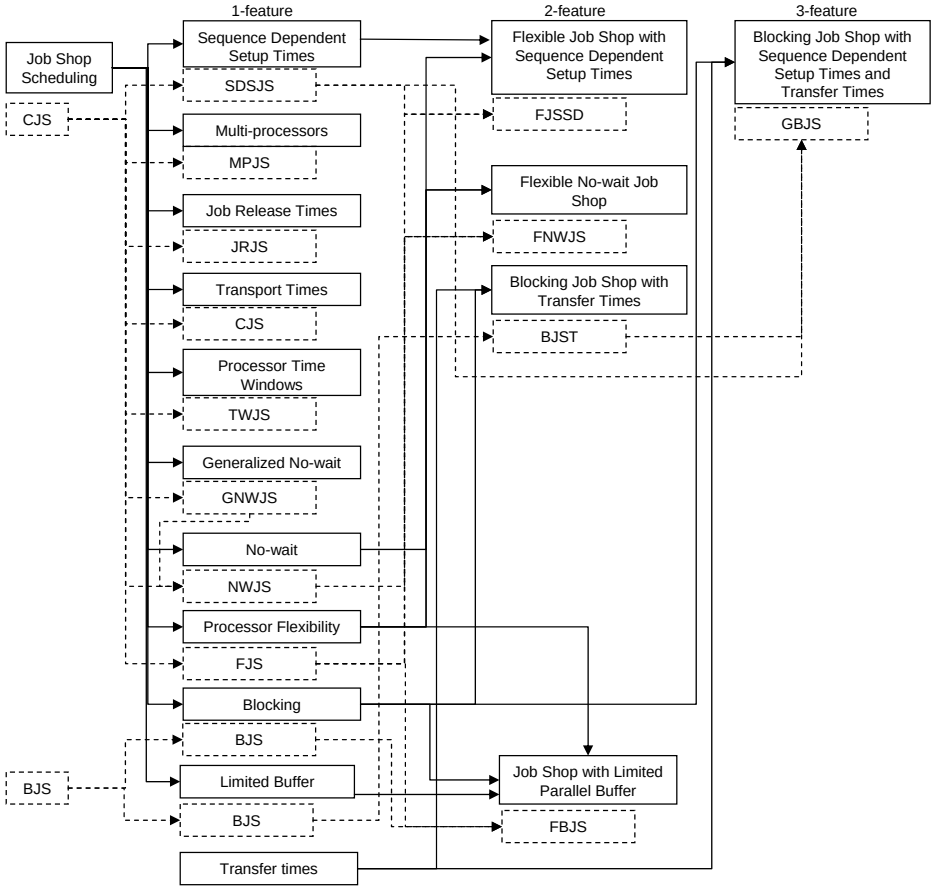

Figure 2-12: Structure tree for complex job shop scheduling problems.

2.5.3 Evaluation of the mathematical programming approach by a general-purpose solver

In order to evaluate the performance of the mathematical programming approach to the presented JS extensions, we selected the extensions for which standard instances have been established. Moreover, the benchmark instances should be accessible for comparison purposes. Our computational experiments include the following selected JS extensions together with their test instances:

1. the FJS with test instances taken from Hurink, Jurisch and Thole (1994) [HJT94],
2. the BJS with test instances taken from Mascis and Pacieralli (2001) [MP02],
3. the NWJS with test instances taken from Mascis and Pacieralli (2001) [MP02].

All MILP formulations were coded in the LPL modeling language version 4.99j and solved by by general-purpose MILP solver CPLEX version 9.0. The original test instances for the FJS in

[HJT94] include 120 instances, which were 40 job shop instances of Lawrence modified by three levels of flexibility. Since the computing times of a Tabu search for instances in the subgroups of sizes 15×10 (i.e. 15 jobs and 10 processors), 20×10 , 30×10 and 15×15 took very long time without achieving proven optimality [HJT94], we dropped these instances from our test and only tested the FJS formulation on 60 instances of sizes $10 \times 5, 15 \times 5, 20 \times 5$ and 10×10. The maximum computing time for those instances reported in [HJT94] is slightly more than 14 minutes; hence we set the computing time limit for all instances at 15 minutes. Mascis and Pacieralli tested their BnB approaches for the BJS and the NWJS on 18 instances of identical size 10×10. As computing times on these instances were rather long (up to 13,255 seconds), we did not use a common time limit for all instances but let each of them run for the time equal to its optimality time in [MP02]. Table 2-11 shows average relative errors and gaps for the BJS and the NWJS. The FJS' computational results for both formulations FJS-1 and FJS-2 are displayed in Table 2-12.

Instance	Size	BJS						NWJS					
	nxm	Opt	Time (s)	UB	LB	RE	Gap	Opt	Time (s)	UB	LB	RE	Gap
abz05	10x10	1641	4669.82	N/F	1067	N/A	34.96	2150	769.51	2493	1509	20.90	39.06
abz06	10x10	1249	3071.10	1864	904	49.24	27.62	1718	92.82	2039	1268	25.70	36.03
ft10	10x10	1158	6215.53	1314	829	13.47	28.41	1607	194.65	1782	1087	15.11	44.91
la16	10x10	1148	5531.21	1555	843	35.45	26.57	1575	124.46	1727	1207	13.24	32.06
la17	10x10	968	2087.34	1305	730.7	34.81	24.51	1371	139.01	2141	832.5	79.55	55.63
la18	10x10	1077	3748.17	1297	792.4	20.43	26.43	1417	196.42	1810	1052	36.49	33.89
la19	10x10	1102	2366.90	1404	791	27.40	28.22	1482	371.18	1645	1046	14.79	39.56
la20	10x10	1118	1471.40	1389	852	24.24	23.79	1526	177.52	1526	1127	0.00	35.69
orb01	10x10	1256	8311.11	1503	843	19.67	32.88	1615	61.35	1722	1077	8.52	42.83
orb02	10x10	1144	498.11	1684	813.9	47.20	28.86	1485	169.34	1893	1036	35.66	39.25
orb03	10x10	1311	569.85	1922	809.6	46.61	38.25	1599	156.59	1769	990	12.97	46.45
orb04	10x10	1246	3254.72	1785	892	43.26	28.41	1653	238.87	1923	1135	21.67	41.54
orb05	10x10	1203	796.31	1601	747.1	33.08	37.90	1365	1025.7	1567	991.5	16.79	31.05
orb06	10x10	1266	12183.68	1525	899.00	20.46	28.99	1555	30.43	1918	1056	28.67	39.42
orb08	10x10	1139	1033.75	1413	722	24.06	36.61	1319	197.51	1466	872.4	12.91	39.21
orb09	10x10	1130	879.42	1405	801	24.34	29.12	1445	62.62	1812	1052	32.48	34.78
orb10	10x10	1367	353.17	1876	817.7	37.23	40.18	1557	843.33	1749	1162	14.05	28.90
Average (%)						31.31	30.69					22.91	38.84

Opt = optimal makespan, UB = upper bound (makespan obtained by CPLEX)
LB = lower bound obtained by CPLEX, N/F = not found, N/A = not applicable
RE = relative error = $(UB - Opt)/Opt$, Gap = $(LB - Opt)/Opt$

Table 2-11: Performance of the MILP approach for on instances of the BJS and the NWJS.

Instance	Size			FJS-1				FJS-2			
	nxm	LB		UB	LB	RE	Gap	UB	LB	RE	Gap
la01-edata	10x5	609	*	609	547	0.00	10.18	609	593	0.00	2.63
la02-edata	10x5	655	*	655	636	0.00	2.90	655	655	0.00	0.00
la03-edata	10x5	550	*	551	481	0.18	12.55	550	547	0.00	0.55
la04-edata	10x5	568	*	568	568	0.00	0.00	568	568	0.00	0.00
la05-edata	10x5	503	*	503	420	0.00	16.50	503	503	0.00	0.00
la06-edata	15x5	833	*	864	493.6	3.72	40.74	833	584	0.00	29.89
la07-edata	15x5	762	*	803	425.7	5.38	44.13	777	505.4	1.97	33.68
la08-edata	15x5	845	*	846	492	0.12	41.78	845	574	0.00	32.07
la09-edata	15x5	878	*	906	491	3.19	44.08	878	554	0.00	36.90
la10-edata	15x5	866	*	885	535.2	2.19	38.20	866	607.7	0.00	29.83
la11-edata	20x5	1087	*	1178	492	8.37	54.74	1172	516	7.82	52.53
la12-edata	20x5	960	*	999	450	4.06	53.13	1001	475	4.27	50.52
la13-edata	20x5	1053	*	1121	487	6.46	53.75	1082	527	2.75	49.95
la14-edata	20x5	1123	*	1171	525	4.27	53.25	1146	543.6	2.05	51.59
la15-edata	20x5	1111	*	1259	493	13.32	55.63	1172	530	5.49	52.30
la16-edata	10x10	892	*	892	887	0.00	0.56	892	892	0.00	0.00
la17-edata	10x10	707	*	707	707	0.00	0.00	707	707	0.00	0.00
la18-edata	10x10	842	*	843	803	0.12	4.63	842	842	0.00	0.00
la19-edata	10x10	796	*	796	796	0.00	0.00	796	798	0.00	-0.25
la20-edata	10x10	857	*	864	839.5	0.82	2.04	857	885	0.00	-3.27
la01-rdata	10x5	570		588	413	3.16	27.54	588	413	3.16	27.54
la02-rdata	10x5	529		543	396	2.65	25.14	541	403	2.27	23.82
la03-rdata	10x5	477		482	349	1.05	26.83	488	349	2.31	26.83
la04-rdata	10x5	502		525	369	4.58	26.49	521	369	3.78	26.49
la05-rdata	10x5	457		486	380	6.35	16.85	471	380	3.06	16.85
la06-rdata	15x5	799		858	413	7.38	48.31	835	413	4.51	48.31
la07-rdata	15x5	749		775	387	3.47	48.33	801	387	6.94	48.33
la08-rdata	15x5	765		799	372	4.44	51.37	820	372	7.19	51.37
la09-rdata	15x5	853		886	467	3.87	45.25	913	436	7.03	48.89
la10-rdata	15x5	804		827	443	2.86	44.90	836	443	3.98	44.90
la11-rdata	20x5	1071	*	1120	478	4.58	55.37	1421	445	32.68	58.45
la12-rdata	20x5	936		969	408	3.53	56.41	1052	408	12.39	56.41
la13-rdata	20x5	1038	*	1090	447	5.01	56.94	1191	447	14.74	56.94
la14-rdata	20x5	1070	*	1124	443	5.05	58.60	1215	443	13.55	58.60
la15-rdata	20x5	1089		1195	473	9.73	56.57	1261	462	15.79	57.58
la16-rdata	10x10	717	*	793	717	10.60	0.00	780	717	8.79	0.00
la17-rdata	10x10	646	*	665	646	2.94	0.00	677	646	4.80	0.00
la18-rdata	10x10	666	*	758	663	13.81	0.45	735	663	10.36	0.45
la19-rdata	10x10	647		753	644	16.38	0.46	760	642	17.47	0.77
la20-rdata	10x10	756	*	814	756	7.67	0.00	775	756	2.51	0.00
la01-vdata	10x5	570	*	589	413	3.33	27.54	587	413	2.98	27.54
la02-vdata	10x5	529	*	540	394	2.08	25.52	560	394	5.86	25.52
la03-vdata	10x5	477		486	349	1.89	26.83	496	349	3.98	26.83
la04-vdata	10x5	502	*	519	369	3.39	26.49	510	369	1.59	26.49
la05-vdata	10x5	457		464	380	1.53	16.85	474	380	3.72	16.85
la06-vdata	15x5	799	*	856	441	7.13	44.81	950	441	18.90	44.81
la07-vdata	15x5	749		776	376	3.60	49.80	917	377.5	22.43	49.60
la08-vdata	15x5	765		793	369	3.66	51.76	1018	369	33.07	51.76
la09-vdata	15x5	853		889	382	4.22	55.22	1097	382	28.60	55.22
la10-vdata	15x5	804	*	875	443	8.83	44.90	915	443	13.81	44.90
la11-vdata	20x5	1071	*	1157	413	8.03	61.44	1555	413	45.19	61.44
la12-vdata	20x5	936	*	1035	408	10.58	56.41	1279	408	36.65	56.41
la13-vdata	20x5	1038	*	1106	382	6.55	63.20	1547	382	49.04	63.20
la14-vdata	20x5	1070	*	1108	443	3.55	58.60	1487	443	38.97	58.60
la15-vdata	20x5	1089		1183	378	8.63	65.29	1748	378	60.51	65.29
la16-vdata	10x10	717	*	831	717	15.90	0.00	1248	717	74.06	0.00
la17-vdata	10x10	646	*	733	646	13.47	0.00	1052	646	62.85	0.00
la18-vdata	10x10	663	*	724	663	9.20	0.00	980	663	47.81	0.00
la19-vdata	10x10	617	*	713	617	15.56	0.00	864	617	40.03	0.00
la20-vdata	10x10	756	*	846	756	11.90	0.00	1207	756	59.66	0.00
Average						5.24	30.82			14.19	29.10

LB = lower bound obtained by CPLEX
RE = relative error = $(UB - BestLB)/BestLB$
Gap = $(LB - BestLB)/BestLB$

Table 2-12: Performance of the MILP approach on instances of the FJS.

The overall performance of the mathematical programming approach was not impressive. While the upper bound quality varied from acceptable (average relative error around 5% for the FJS instances) to low (average relative error around 30% for the BJS and NWJS instances), the lower bound quality was low (average gap around 30%) for all JS extensions in our study. This agrees with the findings in the computational experiments on the CJS formulation's performance with standard instances in Section 2.4. Similar experiments by ILOG on other CJS instances also attest that the upper bound obtained by CPLEX is quite close to the optimum, but the lower bound is just of 67% of the optimum [DRLP03]. Amtatürk and Savelsberch suggested that the computational difficulty with big-M based formulations is that the LP relaxation often gives a fractional of $y_{ij\text{-value}}$ even if the disjunctive statement $x_j - x_i \geq p_i$ or $x_i - x_j \geq p_j$ is satisfied, which leads to unnecessary branchings [AS05]. Our initiatives to tighten these constraints did not bring the hoped results, possibly because decreasing the coefficient of H comes with the increase in the number of binary variables and the number of constraints. All together, the use of MILP formulations together with general-purpose solvers is limited to scheduling problem sizes often too small to be of use in practice.

2.6 Overview of other solution approaches to job shop related problems

A scheduling problem is called job shop related if it is based on the JS with possibly one or several complexifying real-life features presented in Section 2.2. The previous section presented a mathematical programming approach to several JS related problems by formulating them as MILP formulations and solve the problems with a general-purpose solver (e.g. CPLEX). This section briefly outlines other solution methods that have been deployed to solve job shop related problems. For a complete treatment of solution methods to the JS, readers are referred to excellent surveys by Blazewicz et al. (1996) [BDP96], Vaessens et al. (1996) [VAL96], and Jain and Meeran (1999) [JM99]. Descriptions of methods to solve the JS with sequence-setup times can be found in the survey by Allahverdi et al. [ANCK06]. There are no equivalent comprehensive surveys for other JS extensions.

In general, solution methods to a job shop related problem can be either exact or approximative, the latter can be further divided into constructive and iterative heuristics.

2.6.1 Exact methods

Branch and Bound (BnB) algorithms employed in general-purpose MILP solvers like CPLEX often use the LP relaxation framework, where branching in a node of the BnB tree corresponds to fixing a specific sequencing variable to value zero or one. Various cutting planes could be added during the process in order to obtain good lower bounds, e.g. cuts used by Applegate and

Cook (19991) to solve the JS [AC91]. Ad hoc BnB algorithms usually utilize specific knowledge of the specific job shop related problem under study. These algorithms are associated with the corresponding disjunctive graphs. An ad hoc BnB algorithm starts with an empty disjunctive arc selection. Branching in the BnB tree then corresponds to an addition of a disjunctive arc to a partial selection. A lower bound is calculated at each node by solving a subproblem, e.g. a one-machine scheduling problem with release dates and due dates. The main shortcoming of all BnB algorithms is their lack of tight lower bounds to prune unpromising branches of the enumeration tree as early as possible, which makes them inadequate to solve job shop instances of real-life size.

2.6.2 Constructive heuristics

A constructive heuristic builds up a feasible solution from the given input data. Constructive heuristics include priority dispatch rules, insertion heuristics, and shifting bottleneck heuristics.

Priority dispatching rules are widely used in practice because of their simplicity and low computing effort, but their results are often of poor quality. Moreover, they can fail to find feasible solutions for highly constrained job shop problems such as the blocking job shop. The common scheme to apply a priority dispatching rule is as follows. At the start, all operations are unscheduled. In a generic step, schedule an unscheduled operation as follows: find the earliest time at which one or several unscheduled operations can start. Among these operations, choose one operation of the highest priority according to some priority rule, and schedule it. Repeat until all operations are scheduled. Examples of priority rules are SPT (shortest processing time), LPT (longest processing time), FCFS (first-come-first-served), etc.

Insertion heuristics are usually based on operation-insertion or job-insertion principles. An operation-insertion algorithm is based on the insertion of an individual operation into a given partial schedule. One version of such an algorithm is to use optimal insertion, i.e. to insert an operation minimizing the objective function for the corresponding partial schedule. A job-insertion algorithm inserts, in a generic step, all operations of a job into a partial schedule. Job insertion for the JS is described in [KH03]; job insertion for various JS extensions is discussed in [GKP08] and [GK07].

Shifting bottleneck heuristics [ABZ88] schedule the operations by repeatedly solving one-processor scheduling problems where each operation on the processor has, besides its processor duration, an earliest starting time (head) and a sojourn time after processing (tail). The choice of which (one-) processor (problem) to consider is done by a simple bottleneck criterion. In a first phase, a sequence of one-processor-problems build up a solution. In a second phase, it improves the solution by repeatedly rescheduling operations on a processor (in that sense, strictly speaking, only the first phase is a constructive heuristic). Shifting bottleneck heuristics often perform rather well at inexpensive computing cost. Initially developed for the JS, the algorithms can be applied to some JS extensions with appropriate modifications made to the disjunctive graph

[WR90][Sch98].

In general, constructive heuristics typically are easy to implement and have low computational costs. Their shortcomings are that they find solutions of often average quality only. Moreover, as other heuristics, they do not provide a quality guaranty in form of a lower bound on the minimum makespan.

2.6.3 Iterative heuristics

Iterative heuristics start from one or more feasible solutions and use the solution(s) as the base(s) to find a better solution. These heuristics can involve local or population search.

Iterative heuristics with local search find a new solution (neighbor) in the neighborhood of a feasible one. The most decisive factor in designing a local search heuristic is the neighborhood function, which is (hopefully) optimum connected and able to easily generate feasible solutions. The most popular local search heuristics are simulated annealing [VLAL92], tabu search [GL97], and variable neighborhood search [HM01]. In *simulated annealing*, if the difference δ between the current schedule's objective function and the one of its neighbor is negative, the neighbor can still be accepted if $\exp(-\delta/t)$, where t is a time-dependent control parameter called "temperature", is greater than a random number in (0,1) generated by a random mechanism. *Tabu search* heuristics are local search method with a mechanism of tabu list that aims to forbid the short-term return to a previously found solution. Some solution of lower quality might be accepted during the search in order to help the search escape a local optimum. *Variable neighborhood search* employs a systematic use of different neighborhoods when several different neighborhoods are available.

Iterative heuristics with population search find a new solution from a pool of feasible solutions called "population". The population evolves according to certain self-adaptation and combination operators. A selected solution would be the best fit member of the population. The most representative population search heuristic is *genetic algorithms*, which create a new solution (child) from two existing solutions (parents) by a combination operator called cross-over; or from an existing solution by a self-adaptation operator called *mutation*.

Research on iterative heuristics with local and population search has been very active for the past two decades. For guidelines to apply local and population search algorithms, refer to Hertz and Widmer (2003) [HW03]. More details on the algorithms can be found in [Ree93]. Iterative heuristics are perhaps the most used tools in solving job shop related problems. According to recent methodology comparisons for the JS [JM99][VAL96], the Tabu search algorithm by Nowicki and Smutnicki [NS96] and a hybrid of Tabu search and Shifting Bottleneck procedure called *Guided Local Search with Shifting Bottleneck* by Balas and Vazacopulos [BV98] are the two best performing heuristics. Similar comparative studies for other extended JS problems are not available.

2.7 Concluding remarks

This chapter studied practical issues in job shop scheduling. Several complexifying features in practice that invalidate some of the JS' assumptions have been systematically presented in this chapter. The chapter evaluated four existing MILP formulations for the JS: the Manne, the Liao-You, the Wagner, and the Wilson formulations, together with a newly proposed Adaptive Manne formulation. From our comprehensive computational experiments, the Manne formulation has proven to be the best, not the Liao-You one as claimed elsewhere. For each of the presented complexifying features, an MILP formulation for the corresponding 1-feature JS extension has been presented. Several of the presented formulations are novel: (i) the formulation for the BJS, (ii) the formulation for the GNWJS, and (iii) the formulation FJS-2 for the FJS. Using the 1-feature formulations as building blocks, we developed a new formulation for the multimode job shop problem, a JS extension with two complexifying features, namely processor flexibility and multiprocessor requirements. Our computational experiments showed that with the current MILP formulations, the performance of available general-purpose MILP solvers (e.g. CPLEX) still has to be much improved, especially in terms of the lower bound quality.

Chapter 3

Flexible generalized blocking job shop problem

3.1 Introduction

After having examined various job shop scheduling problems with additional features that occur in practice, having modeled these problems as MILP, and also having seen the limitations in solving these MILP, we address in this chapter a complex job shop scheduling problem with *four* additional features namely *blocking constraints, flexibility, sequence-dependent setup times,* and *job transfer times.* We shall refer to this problem as the *Flexible Generalized Blocking Job Shop (FGBJS).* The problem has not yet been addressed in the literature to our knowledge, although it can be used to model a variety of real world scheduling problems. After formulating the problem, we shall develop several heuristics based on novel neighborhoods and tabu search, and present extensive computational results.

The FGBJS extends the version of the scheduling problem without flexibility which has been addressed in Klinkert's dissertation [Kli01] and a paper [GK05] as the so-called *Generalized Blocking Job Shop* (*GBJS*). In [GK05], Gröflin and Klinkert solve the GBJS in a tabu search approach, using a neighborhood based on job insertion that allows them to consistently generate *feasible* solutions and achieve good solution quality. In this chapter, we shall borrow or extend several ingredients from this previous work on GBJS while formulating and developing solution approaches for the FGBJS. In addition, we shall adapt some of recent research findings on job insertion in non-flexible generalized job shop scheduling in [GK07] to the FGBJS. Also, for our computational experiments, we have benefited form the software used in [Kli01] and [GK05], which we have modified for the FGBJS.

The chapter is structured as follows. The next section informally introduces the FGBJS and illustrates it with some applications. Section 3.3 provides a formal problem formulation as a MILP. Section 3.4 presents results from [Kli01], [GK05], and [GK07] for the GBJS, which are

to be extended for the FGBJS. Section 3.5 develops a graph presentation with local operation flexibility, which extends the disjunctive graph representation for the GBJS. Section 3.6 derives three neighborhood structures for the FGBJS. Based on these neighborhood structures, Section 3.7 develops three constructive heuristics while Section 3.8 presents six tabu search based heuristics. Performance of these heuristics on newly created benchmark instances is presented in Section 3.9. Finally, Section 3.10 concludes the chapter with some remarks.

3.2 The FGBJS and some applications

The FGBJS is informally described as follows. A number of jobs have to be processed by a number of processors. All jobs and processors are available continuously from time zero. There is no buffer at or between the processors. Each job consists of a chain of operations to be processed in that order. Each operation of any job comprises four consecutive steps: (1) a *take-over* step where the job is taken from its previous processor, or the job is loaded on a processor if the processor is to perform the job's first operation, (2) a *processing* step where the job is processed for a positive duration, (3) a *waiting* step where the job is *waiting* for an unknown duration (possibly zero) for its next processor to become available and thus *blocking* its current processor, and (4) a *hand-over* step where the job is handed over to its next processor or unloaded from the current one if the operation is the job's last operation. Each and every step takes place without preemption. Each job transfer has its hand-over and take-over steps synchronized; these two steps start and (typically) end at the same time. Each operation can be processed by one processor selected from several alternative processors, and the operation's processing time might depend on the processor assigned to it. Any processor can perform only one operation at a time. There is a *sequence-dependent setup* between two operations consecutively performed on the same processor. Besides sequence-dependent setup times, there exist for any processor's operation sequence a *first setup* time to prepare the first operation on the processor and a *last setup* time to clear the processor after its last operation is finished. All setup times for an operation can be processor-dependent.

Any feasible schedule determines one processor and a starting time for each operation such that the sequence of operations of each job is maintained and at any time, no processor is used by more than one operation. The objective is to find a feasible schedule that minimizes the time to finish all jobs, which is referred to as the *makespan*.

Clearly, the FGBJS provides flexibility in modeling scheduling problems in practice to a larger extent than the extended JS models described in Chapter 2 does, because it combines more practical aspects of complex scheduling. In fact, the scheduling examples mentioned in Chapter 1 can be modeled as a FGBJS as follows.

1. *Example 1.1 (robot cell scheduling)*: Each part is modeled as a job while each CNC machine or the transport robot as a processor. For any part, its processing on a machine is preceded

and followed by two moving operations by the robot. Flexibility is evident as a part can be processed by any machine equipped with the right tool. Synchronization takes place between the robot and a machine when the robot feeds the machine with a part or remove the part from it. Job transfer times are often assumed to be negligible. Blocking happens when (1) a part stays on the robot until its next assigned machine becomes free, or (2) a part stays on a machine waiting for the robot to come. The duration of the robot's travel without holding any part from a CNC machine to another one can be modeled as a sequence-dependent setup time.

2. *Example 1.2 (automated high-density warehouse scheduling):* Each pallet retrieval can be modeled as a job consisting of three consecutive operations: (1) the pallet's retrieval from a storage position by a carriage integrated into the aisle holding the pallet, (2) the pallet's move along a cross-aisle by its integrated carriage, and (3) the pallet's carrying by an elevator. Pallet routing flexibility may exist, for instance when elevators are installed at both ends of a cross-aisle. Blocking occurs when a pallet is waiting on a carriage for its next carriage or elevator to arrive. Travelling time for an empty carriage depends on positions of two pallets to be retrieved consecutively, so it is sequence-dependent. It may also depend on the carriage, thus it is processor-dependent. Hand-over and take-over steps involved in any pallet transfer normally take some positive time. For a fixed number of pallets, maximizing the throughput of the warehouse is equivalent to minimizing the time to finish moving all parts, i.e. the makespan.

3. *Example 1.3 (surgical case scheduling):* This problem can be modeled a FGBJS problem in which patients are modeled as jobs while the most expensive resource in each of the three phases (e.g. nurses in the preoperative phase, operating rooms in the perioperative phase, and recovery beds in the postoperative phase) are modeled as processors. There is flexibility in choosing a resource for a patient in each phase, e.g. the patient can be operated in one of alternative operating rooms. Turnover time between two consecutive operations in an operating room depends on the first operation's cleanup time and the second operation's setup time, hence it is sequence-dependent. When a patient cannot go on to the next phase if a resource over there is not yet available, it occupies and thus blocks the current phase's resource while waiting for the next phase's resource. Note that transport and transfer times are often negligible when transporters are always available and holding units, operating rooms, and recovery rooms are in proximity of one another.

4. *Example 1.4 (railway scheduling):* Each block section can be modeled as a processor, each train's entire travel as a job, and each train's traversal through a block section as an operation. The start of each operation is the time when the head of the train enters the section and the operation's completion time is the time the head enters the next section. Job transfer times are the synchronization times between two consecutive section traversals, hence they are zero. Setup time for an operation is the time it takes

a preceding train to completely leave the block, plus some safety time. This time is sequence-dependent since it depends on the previous train's length and speed. Once a train has to stay in a section for some reason, e.g. because there is an accident in the next section, no other train can enter the section and the section is thus blocked. A train can take one of several parallel segments in a station or be rerouted to deal with unexpected events, which implies flexibility.

3.3 Problem formulation

This section formally formulates the FGBJS as a MILP problem. Let $\mathcal{J}$ be a set of n jobs, M a set of m processors, and I be a set of o operations. A job $J \in \mathcal{J}$ is a set of operations to be processed in some order. Therefore, $J \subseteq I$, and $\mathcal{J}$ is a partition of I, i.e. $\mathcal{J} \subseteq 2^I$. For any job J, index its operations in the job's processing order as $J_1, \ldots, J_{|J|}$. Let the job of $i \in I$ be J^i. Define a set of pairs of consecutive (hence ordered) operations for job J as $A_J = \{(i,j) : J^i = J^j = J, i = J_r, j = J_{r+1}, 1 \leq r < |J|\}$. For each $(i,j) \in A_J$, i is the immediate job predecessor of j, we denote $i = JP(j)$, and j is the immediate job successor of $i, j = JS(i)$. Define σ and τ as two dummy operations of zero duration where σ starts before and τ starts after all other operations respectively. Associated with each operation $i \in I$ is a set of assignable processors $M_i \subseteq M$; each processor $k \in M_i$ processes i for a duration p_{ik}. Each operation is performed by the one processor assigned to it, and each processor can perform only one operation at a time. Let μ be a complete assignment that assigns each operation $i \in I$ to a processor $k \in M_i$, i.e. $\mu(i) := k$. Denote by I_k the set of operations assigned to processor k under μ. Operation $i \in I$ is composed of four steps: (1) a take-over step t_i of duration p_i^t, (2) a processing step for a processor-dependent duration p_{ik} ($k = \mu(i)$), (3) a waiting step for an unknown duration, and (4) a hand-over step h_i of duration p_i^h. The hand-over step of an operation and the take-over step of the operation's immediate job successor are synchronized. Denote by $p^s_{\sigma ik}$ and $p^s_{i\tau k}$ respectively the setup times for operation i as first and the last operation on processor $k \in M_i$, and let p^s_{ijk} be a setup time for two operations $i, j \in I$, to be consecutively performed on $k \in M_i \cap M_j \neq \emptyset$. No preemption is allowed for any operation. All processing times are positive; other durations are nonnegative. All jobs and processors are continuously available from time zero. The objective is to find a feasible schedule that respects the processing order of each job and the capacity of each processor, and minimizes the makespan.

Example 3.1.

Consider a FGBJS instance of 3 jobs $\mathcal{J} = \{1,2,3\}$, 3 processors $M = \{1,2,3\}$, and 7 operations $I = \{1,2,...,7\}$. The operations are partitioned into jobs and listed in the jobs' processing order as follows: job $1 = \{1,2\}$, job $2 = \{3,4,5\}$, and job $3 = \{6,7\}$. Sets of consecutive operations for the jobs are $A_1 = \{(1,2)\}$, $A_2 = \{(3,4),(4,5)\}$, and $A_3 = \{(6,7)\}$. Sets of assignable processors for operations are $M_1 = \{1\}$, $M_2 = \{2\}$, $M_3 = \{1,3\}$, $M_4 = \{2\}$, $M_5 = \{3\}$, $M_6 = \{1,2\}$, and

$M_7 = \{3\}$. Processor-dependent processing times are given in Table 3-1. All take-over and hand-over times, as well as setup times for the first and last operations on any processor, are set equal to 10. Table 3-2 shows the sequence-dependent setup times. Setup times are assumed to be processor-independent for the sake of simplicity.

Processor/Operation	1	2	3	4	5	6	7
1	20		20			20	
2		20		20		20	
3			30		20		20

Table 3-1: Processing times in Example 3.1

Operation/Operation	1	2	3	4	5	6	7
1			10			10	
2				10		10	
3	20				10	10	10
4		10				10	
5							10
6	10	10	20	10			
7			10		10		

Table 3-2: Sequence dependent setup times in Example 3.1.

The Gantt chart of a feasible schedule is shown in Figure 3-1 (the chart labels the operations in the format *Job number.Index of operation* (*J.r*) instead of the subscripted format J_r to fit the font size to the boxes, e.g. 2.3 refers to the third operation of job 2). In this schedule, operation 3 (2.1) is assigned to processor 1, operation 6 (3.1) to processor 2, and each of the other operations is assigned to its single processor. Observe the synchronization of two consecutive operations of any job, e.g. the hand-over step of operation 6 (3.1) and the take-over step of operation 7 (3.2) both start at time 40 and end at time 50. Observe further that although operation 3 (2.1) is completed at time 40, job 2 cannot be moved to processor 2 since this processor is still processing job 3. Therefore job 2 has to stay on processor 1 and thus blocks this processor from processing job 1 until time 60 when processor 2 becomes available, and transfer of job 2 occurs. The resulting makespan is 170. □

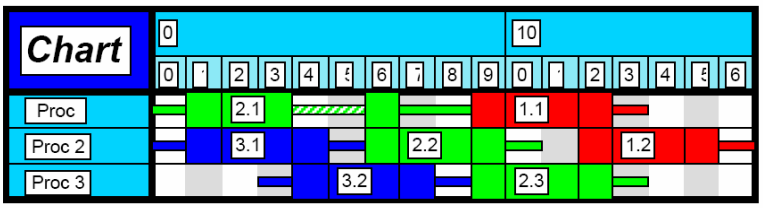

Figure 3-1: Gantt chart in Example 3.1.

In order to give a mixed integer linear programming (MILP) formulation for the FGBJS, define the decision variables as follows. Let x_τ be the starting time of dummy operation τ, x_i^t and x_i^h

be the starting times of the take-over and the hand-over steps of operation $i \in I$ respectively. Define assignment binary variables z_{ik} as:

$$z_{ik} = \begin{cases} 1 \text{ if } i \text{ is assigned to } k \in M_i \\ 0 \text{ otherwise} \end{cases}.$$

Also for two operations $i, j \in I, i < j$, $M_i \cap M_j \neq \emptyset$, define sequence variable y_{ij} as:

$$y_{ij} = \begin{cases} 1 \text{ if } i \text{ precedes } j \text{ (not necessarily immediately) on some } k \in M_i \cap M_j \\ 0 \text{ otherwise} \end{cases}.$$

All notations are summarized in Table 3-3. A MILP formulation for the FGBJS is given below:

Notations	
Sets	
$\mathcal{J}$	Set of jobs J
I	Set of operations
M	Set of processors
M_i	Set of processors assignable to $i \in I$
A_J	Set of pairs of consecutive operations $(J_r, J_{r+1}), 1 \leq r < \lvert J\rvert$ of job $J \in \mathcal{J}$
Parameters	
p_{ik}	Processing time of operation $i \in I$ on processor $k \in M_i$
p_i^t	Take-over time of operation $i \in I$
p_i^h	Hand-over time of operation $i \in I$
$p_{\sigma ik}^s$	Setup time of operation $i \in I$ as the first operation on processor $k \in M_i$
$p_{i\tau k}^s$	Setup time of operation $i \in I$ as the last operation on processor $k \in M_i$
p_{ijk}^s	Setup time between $i, j \in I$ on $k \in M_i \cap M_j \neq \emptyset$
H	Very large positive number
Decision variables	
$z_{ik} =$	$\begin{cases} 1 \text{ if } i \in I \text{ is assigned to processor } k \in M_i \\ 0 \text{ otherwise} \end{cases}$
$y_{ij} =$	$\begin{cases} 1 \text{ if } i \text{ is sequenced before } j \text{ on some processor } k \in M_i \cap M_j, i, j \in I, i < j \\ 0 \text{ otherwise} \end{cases}$
x_i^t	Starting time of the take-over step t_i of $i \in I$
x_i^h	Starting time of the hand-over step h_i of $i \in I$
x_τ	Starting time of dummy operation τ

Table 3-3: Notations for the MILP.

$$\text{Minimize } x_\tau \text{ subject to :} \tag{3-1}$$

$$x_i^h - x_i^t - \sum_{k \in M_i} p_{ik} z_{ik} \geq p_i^t \quad \text{for all } i = J_1, J \in \mathcal{J} \tag{3-2}$$

$$x_j^h - x_j^t - (p_j^t + p_{jk}) z_{jk} \geq 0 \tag{3-3}$$
$$\text{for all } (i,j) \in A_J, J \in \mathcal{J}, k \in M_j - M_i$$

$$x_j^h - x_j^t - p_{jk} z_{jk} - p_j^t (z_{jk} - z_{ik}) - p_{ijk}^s (z_{ik} + z_{jk} - 1) \geq 0 \tag{3-4}$$
$$\text{for all } (i,j) \in A_J, J \in \mathcal{J}, k \in M_i \cap M_j$$

$$x_i^h - x_j^t = 0 \quad \text{for all } (i,j) \in A_J, J \in \mathcal{J} \tag{3-5}$$

$$x_j^t + H(2 - z_{ik} - z_{jk}) + H(1 - y_{ij}) - x_i^h \geq p_i^h + p_{ijk}^s \tag{3-6}$$
for all $i, j \in I, i < j, (i,j) \notin A_{J \in \mathcal{J}}, k \in M_i \cap M_j \neq \emptyset$

$$x_i^t + H(2 - z_{ik} - z_{jk}) + Hy_{ij} - x_j^h \geq p_j^h + p_{jik}^s \tag{3-7}$$
for all $i, j \in I, i < j, (i,j) \notin A_{J \in \mathcal{J}}, k \in M_i \cap M_j \neq \emptyset$

$$x_i^t \geq \sum_{k \in M_i} p_{\sigma ik}^s z_{ik} \quad \text{for all } i \in I \tag{3-8}$$

$$x_\tau - x_i^h \geq p_i^h + \sum_{k \in M_i} p_{i\tau k}^s z_{ik} \quad \text{for all } i \in I \tag{3-9}$$

$$\sum_{k \in M_i} z_{ik} = 1 \quad \text{for all } i \in I \tag{3-10}$$

$$z_{ik} \in \{0,1\} \quad \text{for all } i \in I, k \in M_i \tag{3-11}$$

$$y_{ij} \in \{0,1\} \quad \text{for all } i, j \in I, i < j, M_i \cap M_j \neq \emptyset \tag{3-12}$$

$$x_i^t, x_i^h \geq 0 \quad \text{for all } i \in I x_\tau \geq 0 \tag{3-13}$$

Objective function (3-1) is to minimize the *makespan*. Constraints (3-2) are *precedence* constraints ensuring that for any operation i which is the first operation of its job, its hand-over step takes place after the completion of its take-over and processing steps. Constraints (3-3) express that if j is on processor k and its job immediate predecessor $i = JP(j)$ is not on k, then $x_j^h - x_j^t \geq p_j^t + p_{jk}$. If both i and $j = JP(i)$ are on k, then the two operations are, in effect, merged to an operation of processing time $(p_{ik} + p_{ijk}^s + p_{jk})$, and constraints (3-4) enforce that $x_j^h - x_j^t \geq p_{ijk}^s + p_{jk}$. *Synchronization* constraints (3-5) make sure that the hand-over step of operation i is synchronized with the take-over step of its immediate job successor $j = JS(i)$. Constraints (3-6) and (3-7) are *disjunctive* constraints ensuring that if i and j are on k then either i is before j and $x_j^t - x_i^h \geq p_i^h + p_{ijk}^s$ or j is before i and $x_i^t - x_j^h \geq p_j^h + p_{jik}^s$. Note that because of possible blocking, the completion time of $i \in I$ on k, defined as $C_i = x_i^t + p_i^t + p_{ik} z_{ik}$, might not be the time at which i leaves k. Therefore, we use the term *departure time* to mention the point of time when the hand-over step of an operation is done and its job either goes on to the next processing phase or leaves the system. Observe that from constraints (3-6) and (3-7), for any three operations u, v, and $w \in I$ sequenced in this order on processor k, we need to have $p_{uvk}^s + p_{vk} + p_{vwk}^s \geq p_{uwk}^s$ as w is not sequenced immediately after u. This requirement is clearly satisfied if the triangle inequality $p_{uvk}^s + p_{vwk}^s \geq p_{uwk}^s$ is assumed for the setup times. If v is the first operation in the sequence on k, then the triangle inequality is $p_{\sigma vk}^s + p_{vwk}^s \geq p_{\sigma wk}^s$. Similarly, if v is the last operation in the sequence on k, then the triangle inequality is $p_{uvk}^s + p_{v\tau k}^s \geq p_{u\tau k}^s$. Constraints (3-8) and (3-9) take into account initial and last setup times for all sequences. *Assignment* constraints (3-10) require that exactly one processor is assigned to each operation. Domains of decision variables are stated in the last four constraints (3-11)-(3-4).

Note that when two sets of assignable processors for any two consecutive operations i and j of the same job are disjoint, or when it is practically impossible to process i and j on the same processor k (e.g. when k should be calibrated off-load after each operation), constraints (3-2),

(3-3), and (3-4) can be replaced by the following constraints

$$x_i^h - x_i^t - \sum_{k \in M_i} p_{ik} z_{ik} \geq p_i^t \quad \text{for all } i \in I; \tag{3-14}$$

while the condition $(i, j) \notin A_{J \in \mathcal{J}}$ is dropped from constraints (3-6) and (3-7).

The MILP formulation above can be coded by any mathematical programming language (e.g. LPL) then solved by any commercial MILP solver (e.g. CPLEX). However, it is certainly more difficult to solve than are the formulations for the JS or other single-feature extensions studied in Chapter 2 since it covers more practical constraints. As results of our computational experiments on the job shop related formulations in Chapter 2 suggest, solving the FGBJS by a general-purpose MILP solver is out of reach, at least currently, for large problem sizes as they occur in practice. Therefore, more efficient heuristic solution approaches should be sought.

3.4 The Generalized Blocking Job Shop

The Generalized Blocking Job Shop (GBJS) introduced in [Kli01] and [GK05] is a special version of the FGBJS where each operation can be performed by only one processor. In other words, the FGBJS is an extension of the GBJS in the dimension of processor flexibility. Therefore, we may extend research results in [Kli01] and [GK05] for the FGBJS. Furthermore, recent results on job insertion in a generalized job shop environment in [GK07] will prove very useful to address the FGBJS. This section briefly presents these research backgrounds. Without loss of generality, assume that a setup time between two non-consecutive operations of the same job assigned to the same processor is not greater than the sum of processing times of their job's in-between operations.

3.4.1 Graph representation for the GBJS

As shown in Chapter 2, solving an JS instance is equivalent to finding a feasible selection in the instance's associated disjunctive graph so that the length of the critical path $\sigma - \tau$ is minimized. This approach can be applied to other extended JS problems [Sch98][WR90]. However, the disjunctive graph cannot represent JS extensions involving blocking constraints because selecting a disjunctive arc in a pair is the same as fixing a sequencing variable y_{ij} in capacity constraints (2-3) and (2-4) of the CJS formulation to value 1 (or 0); it is shown in Section 2.5 that this value fixing fails to enforce blocking. Therefore, Klinkert [Kli01] and Gröflin and Klinkert [GK05] proposed a new generalized disjunctive graph in their solution method for the GBJS, which we refer to throughout of this chapter as the *disjunctive blocking graph*.

Reusing all of the FBGJS' notations introduced in Section 3.2 (except that processing times

and setup times are not processor-dependent due to non-flexibility), we define for a given GBJS instance its associated disjunctive blocking graph $G = (V, A, E, \mathcal{E}, c)$ as follows. Each operation $i \in I$ is represented by two nodes t_i and h_i corresponding to the take-over and hand-over steps of the operation. The node set $V = V^I \cup \{\sigma, \tau\}$, where $V^I = \{t_i, h_i : i \in I\}$, and nodes σ and τ respectively represent the dummy start and end operations. The set of conjunctive arcs $A = A^0 \cup A^1 \cup A^{\sigma,\tau}$. Set A^0 is the set of pairs of synchronization arcs joining hand-over node h_i to take-over node t_j and vice versa where i and j are two consecutive operations of the same job, i.e. $A^0 = \{(h_i, t_j)(t_j, h_i) : (i, j) \in A_J, J \in \mathcal{J}\}$. Set A^1 is the set of operational arcs, i.e. $A^1 = \{(t_i, h_i) : i \in I\}$. Finally, $A^{\sigma,\tau} = A^\sigma \cup A^\tau$ where $A^\sigma = \{(\sigma, t_i) : i \in I\}$ is the set of arcs connecting dummy start node σ to each take-over node and $A^\tau = \{(h_i, \tau) : i \in I\}$ is the set of arcs joining the hand-over nodes to dummy end node τ. The set E of disjunctive arcs contains, for each pair of operations $\{i, j\}$ to be performed on k $(i, j \in I_k)$, disjunctive arcs $e = (h_i, t_j)$ and its mate $\overline{e} = (h_j, t_i)$, i.e. $E = \{(h_i, t_j), (h_j, t_i) : i, j \in I_k, k \in M\}$. Denote by $\mathcal{E}$ the family of all disjunctive sets $D = \{e, \overline{e}\}$, we have $\mathcal{E} \subseteq 2^E$ and $E = \bigcup_{D \in \mathcal{E}} D$.

The arc weights c are set as follows: $c_e := 0$ for $e \in A^0$, $c_e := p^s_{\sigma i}$ for $e = (\sigma, t_i) \in A^{\sigma,\tau}$, $c_e := p^h_i + p^s_{i\tau}$ for $e = (h_i, \tau) \in A^{\sigma,\tau}$, $c_e := p^t_i + p_i$ for $e = (t_i, h_i) \in A^1$, and $c_e := p^h_i + p^s_{ij}$ for $e = (h_i, t_j) \in E$.

The disjunctive blocking graph differs from the disjunctive graph for the classical job shop in the following main points: (1) there are two nodes (t_i and h_i) instead of one to represent an operation; (2) each disjunctive set $D = \{(h_i, t_j), (h_j, t_i)\}$ is composed of two disjunctive arcs of different ends instead of two arcs with same ends; and (3) the conjunctive arc set contains pairs of synchronization arcs $\{(h_i, t_j), (t_j, h_i) : (i, j) \in A_J, J \in \mathcal{J}\}$ where each synchronization pair forms a zero cycle.

Note that Mascis and Pacciarelli proposed in [MP02] another graph representation for the Blocking Job Shop (see Section 2.5.1) called the *alternative graph*, which is an extension of the classical disjunctive graph and can also be used to represent the GBJS. However, according to our modeling experience, it is somewhat more difficult to track disjunctive relations among operations performed on the same processor in the alternative graph than in the disjunctive blocking graph since the former expresses a blocking relation of an operation with another one through the representative node of the operation's job successor.

A *selection* S is a subset of the set of disjunctive arcs, $S \subseteq E$. Selecting S means adding a number of disjunctive arcs to the set of conjunctive arcs A, which results in a *subgraph* $G(S) = (V, A \cup S, c)$. S is said to be *complete* if $S \cap D \neq \emptyset$ for all $D = \{e, \overline{e}\} \in \mathcal{E}$. If there is no positive cycle in $G(S)$ then S is *positive acyclic*. Clearly, any acyclic selection S satisfies $|S \cap D| \leq 1$ for all $D \in \mathcal{E}$; otherwise a trivial cycle containing some pair $\{e, \overline{e}\}$ exists. S is *feasible* if it is complete and positive acyclic. The resulting graph $G(S)$ is called the *solution graph* of S. The GBJS problem is equivalent to the problem of determining a feasible selection S in G so that the length of a critical path $L(S)$ in $G(S)$ is minimized. An arc lying on a critical

path is called a *critical arc,* and its head and tail nodes are called *critical nodes.* An operation is said to be a *critical operation* if at least one of its two representing nodes (hand-over and take-over) is critical.

The graph representation for the GBJS is illustrated by the following example.

Example 3.2.

We create a GBJS instance from the FGBJS instance in Example 3.1 by setting $M_3 := \{1\}$ and $M_6 := \{1\}$ while keeping the (single) processors for the other operations. Disjunctive blocking graph $G = (V, A, E, \mathcal{E}, c)$ associated with this GBJS instance has a node set $V = \{t_i, h_i : i = 1, \ldots, 7\} \cup \{\sigma, \tau\}$; a set of constructive arcs $A = A^0 \cup A^1 \cup A^{\sigma,\tau}$ where $A^0 = \{(h_1, t_2), (t_2, h_1), (h_3, t_4), (t_4, h_3), (h_4, t_5), (t_5, h_4), (h_6, t_7), (t_7, h_6)\}$, $A^1 = \{(t_i, h_i) : i = 1, \ldots, 7\}$, $A^{\sigma,\tau} = \{(\sigma, t_i) : i = 1, \ldots, 7\} \cup \{(h_i, \tau) : i = 1, \ldots, 7\}$; a set of disjunctive arcs

$E = \{(h_1, t_3), (h_3, t_1), (h_1, t_6), (h_6, t_1), (h_3, t_6), (h_6, t_3), (h_2, t_4), (h_4, t_2), (h_7, t_5), (h_5, t_7)\}$;

and a family of disjunctive sets containing disjunctive arc e and its mate $\overline{e}$

$\mathcal{E} = \{\{(h_1, t_3), (h_3, t_1)\}, \{(h_1, t_6), (h_6, t_1)\}, \{(h_3, t_6), (h_6, t_3)\}, \{(h_2, t_4), (h_4, t_2)\}, \{(h_7, t_5), (h_5, t_7)\}\}$.

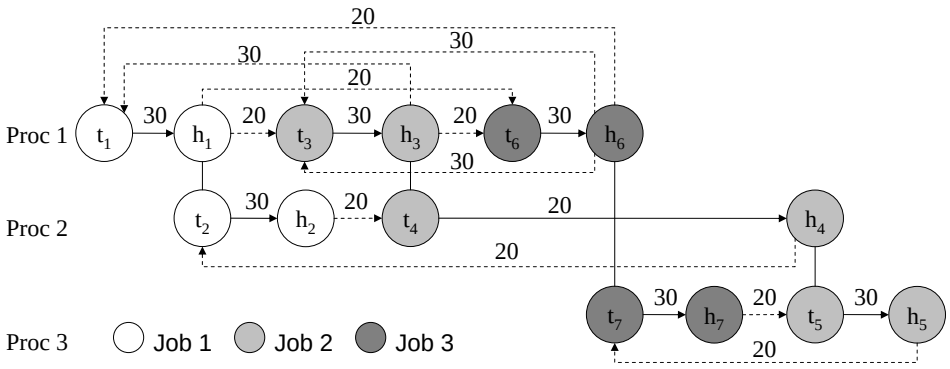

Figure 3-2: Disjunctive blocking graph for Example 3.2.

Figure 3-2 displays disjunctive blocking graph G. Pairs of synchronization arcs with zero weight are drawn as edges and the arcs in $A^{\sigma,\tau}$ are not shown for the sake of clarity.

3.4.2 Job insertion and conflict graph

Job insertion activities frequently occur in practical scheduling, e.g. when updating a schedule upon an arrival of a new job. In fact, there is a class of scheduling problems called *job insertion problems*, where a job needs to be inserted into an existing schedule so that an updated schedule is feasible and its makespan is minimized. For further information on job insertion problems in various job shop environments, the reader is referred to [KH03] (the classical job shop), [GKP08] (the multiprocessor task job shop), and [GK07] (a generalized job shop). This section

recalls some research results on job insertion in a generalized job shop [GK07] (and their proofs if deemed necessary) with slight modifications to adapt them to the GBJS. Our motivation is to use the adapted results to derive a scheme to obtain a feasible solution from a given feasible one in the next subsection.

Given disjunctive blocking graph $G = (V, A, E, \mathcal{E}, c)$, define the node set of job J as $V_J = \{h_u, t_u : u \in J\}$, the set of disjunctive arcs entering J as $E_J^- = \{e \in E : tail(e) \notin V_J, head(e) \in V_J\}$, the set of disjunctive arcs leaving J as $E_J^+ = \{e \in E : tail(e) \in V_J, head(e) \notin V_J\}$, and the set of disjunctive arcs incident to J as $E_J = E_J^- \cup E_J^+$. Let $\mathcal{E}_J \subset \mathcal{E}$ be the family of disjunctive pairs $D = \{e, \overline{e}\}, \cup_{D \in \mathcal{E}_J} = E_J$. Assume that a selection $R \subseteq E - E_J$ such that $|R \cap D| = 1$ for all $D \in \mathcal{E} - \mathcal{E}_J$ and R is positive acyclic has been determined. We need to insert J into the partial schedule of $\mathcal{J} - J$ jobs to obtain a feasible schedule in which a predetermined operation $i \in J$ is sequenced before a predetermined operation j, $j \neq i$ and j is not a job predecessor of i, on processor k. Denote by $G^J = (V, A \cup R, E_J, \mathcal{E}_J, c)$ the disjunctive blocking graph associated with the insertion of job J and call G^J the *(job) insertion graph* of J. A selection in G^J is $S_J \subseteq E_J, S_J$ can also be called an *insertion*. S_J is complete if $S_J \cap D \neq \emptyset$ for all $D \in \mathcal{E}_J$; S_J is positive acyclic if $G^J(S_J) = (V, A \cup R \cup S_J)$ contains no positive cycle; and S_J is feasible if it is complete and positive acyclic. As operation i is required to be before operation j on processor k in any solution, $(h_i, t_j) \in E_J^+$ is thus a *preselection.*

Proposition 1. *Given an insertion graph G^J, if a selection S_J results in a positive cycle Z in $(V, A \cup R \cup S_J)$, then there exists a "short" positive cycle Z' in $(V, A \cup R \cup S_J)$ with $Z' \cap E_J \subseteq Z \cap E_J$ and Z' visits job J once.*

Proof. Suppose there is a positive cycle Z in $G^J(S_J)$. If Z visits J only once, then $Z' = Z$. Otherwise, let q be a times that Z visits J, i.e. $|Z \cap E_J^-| = |Z \cap E_J^+| = q, 1 < q \leq |J|$. Number q entering arcs in the order they enter $J, Z \cap E_J^- = \{e_1, \ldots, e_q\}$. Number q leaving arcs similarly, $Z \cap E_J^+ = \{f_1, \ldots, f_q\}$. Cycle Z can be expressed as a concatenation $Z = \{e_1, P_1, f_1, Q_1, \ldots, e_q, P_q, f_q, Q_q\}$ where for $p = 1, \ldots, q-1$, P_p is a path from $head(e_p)$ to $tail(f_p)$ of positive length as P_p contains at least one operational arc and Q_p is a path from $head(f_p)$ to $tail(e_{p+1})$. By convention, the indices are given modulo q, i.e. $e_{q+1} = e_1$. Let r be a job visiting index associated with the highest indexed operation that must be processed after all other $q-1$ operations in the job processing order of $J, 1 \leq r \leq q$. Consider a path (f_r, Q_r, e_{r+1}) from $tail(f_r)$ to $head(e_{r+1})$ and a path P_{r+1} from $head(e_{r+1})$ to $tail(f_r)$, P_{r+1} exists because operation associated with visiting index $r+1$ is processed before operation associated with visiting index r in the processing of job J; these two paths form together a cycle Z'. Clearly Z' is positive because it contains at least one conjunctive operational arc. Furthermore, Z' visits job J once through entering arc e_{r+1} and leaving arc f_r. □

Cycle Z' is drawn in bold in Figure 3-3. Proposition 1 implies that if a selection in G^J results in no "short" cycle, then this selection is positive acyclic.

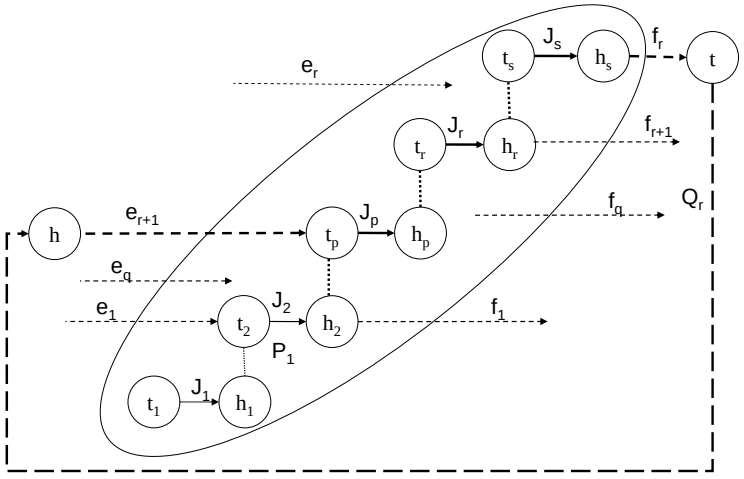

Figure 3-3: A short cycle visiting job J once.

Besides the insertion graph, another graph representation to express insertion of job J when all jobs in $\mathcal{J} - J$ have been scheduled is proposed in [GK07].

Definition 1. *Given an insertion graph $G^J = (V, A \cup R, E_J, \mathcal{E}_J, c)$, the conflict graph of G^J is the undirected graph $H_{G^J} = (E_J, U)$ where for any $e \in E_J^-$ and $f \in E_J^+$, $(e, f) \in U$ if there is a short positive cycle in G^J passing through $\{e, f\}$.*

H_{G^J} is a bipartite graph with two separate node sets, $E_J = E_J^- \cup E_J^+$. Clearly, $(e, \overline{e}) \in U$ for all $e \in E_J$. The concepts of insertion and conflict graphs are illustrated in the following example.

Example 3.3.

Consider the disjunctive blocking graph G from Example 3.2. Suppose jobs 1 and 3 have been scheduled with $R = \{(h_1, t_6)\}$ and job $J = 2$ is to be inserted. Construct G^2 as shown to the left of Figure 3-4 (with arcs in R are drawn in bold and dashed) and its associated conflict graph $H_{G^2} = (E_2, U)$ as shown to the right of Figure 3-4. Besides $(e, \overline{e}) \in U$, edges $(e, f) \in U, f \neq \overline{e}$, are: (1) $(a, \overline{c})$ due to cycle Z_1 in G^2 passing through nodes $(h_1, t_3, h_3, t_4, h_4, t_2)$, (2) $(\overline{a}, c)$ due to $Z_2 = (h_3, t_1, h_1, t_2, h_2, t_4)$, (3) $(b, \overline{d})$ due to $Z_3 = (h_6, t_3, h_3, t_4, h_4, t_5, h_5, t_7)$, and (4) $(\overline{c}, d)$ due to $Z_4 = (h_4, t_2, h_1, t_6, h_6, t_7, h_7, t_5)$. □

Theorem 2. *There is a one-to-one correspondence between the feasible selections in $G^J = (V, A \cup R, E_J)$ and the stable sets of maximum cardinality $|E_J|/2$ in the conflict graph $H_{G^J} = (E_J, U)$.*

Proof. Suppose there is a feasible selection S_J in G^J, then $(V, A \cup R \cup S_J)$ is positive acyclic by definition. Therefore, there is no $\{e, f\} \subseteq S_J$ such that $(e, f) \in U$ and S_J is thus a stable set in H_{G^J}. Since S_J is feasible, it is complete and hence $|S_J| = |E_J^-| = |E_J^+| = |E_J|/2$. Conversely,

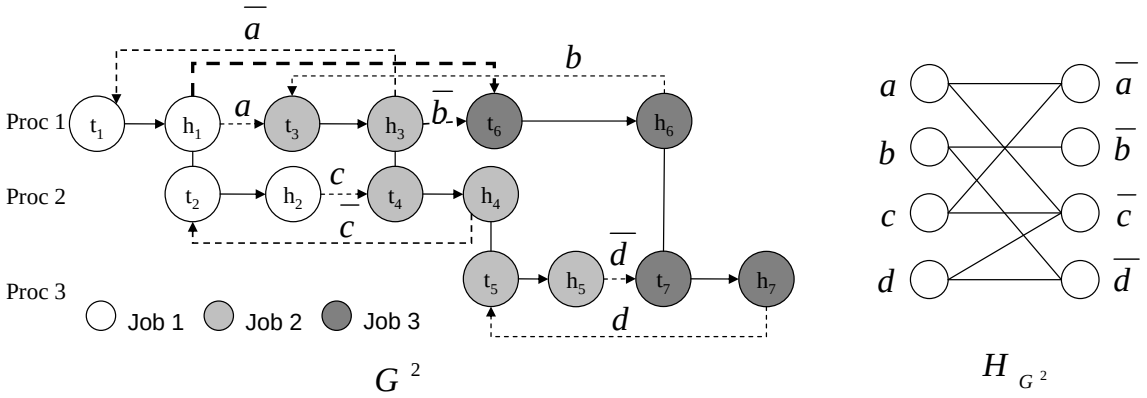

Figure 3-4: Insertion graph G^2 and its associated conflict graph H_{G^2} in Example 3.3.

let S_J be a stable set of maximum cardinality in H_{G^J}, hence $|S_J| = |E_J^-| = |E_J^+| = |E_J|/2$ and S_J is complete. If S_J forms some cycle in $G^J(S_J)$, then $(V, A \cup R \cup S_J)$ contains a short cycle Z' visiting J once according to Proposition 1. Z' enters J through some $e \in S_J^-$ and leaves J through some $f \in S_J^+$. But then $(e, f) \in U$, contradicting to S_J being stable in H_{G^J}. □

The following prerequisite terms are needed for our derivation of a feasible selection in G^J when given a set of preselections (e.g. $\{(h_i, t_j)\}$) in the next subsection.

Definition 2. *Let $H_{G^J} = (E, U)$ be the conflict graph of G^J. Let a sequence of distinct nodes $P = (e_0, f_1, e_1, f_2, e_2, \ldots, f_n, e_n), n \geq 0$, be an alternating path from node e_0 to node e_n if $(e_{k-1}, f_k) \in U, f_k \neq \overline{e}_{k-1}$, and $e_k = \overline{f}_k$, for all $1 \leq k \leq n$, and denote such a node sequence as $e_0 \rightsquigarrow e_n$. Path $P = (e_0, e_0)$ is a trivial alternating path.*

Definition 3. *Let $S_J \subseteq E_J$ be a selection in G^J. The* closure $\Phi(S_J)$ *is $\Phi(S_J) = \{f \in E_J : e \rightsquigarrow f$ for some $e \in S_J\}$.*

Example 3.4.

Consider the insertion graph G^2 and conflict graph H_{G^2} constructed in Example 3.3. Suppose $S_J = \{c\}$, $J = 2$, we have $\Phi(S_J) = \{c, a\}$ as there exists $c \rightsquigarrow a = (c, \overline{a}, a)$ (see the dark nodes and the bold edges in the right graph in Figure 3-5). The selected disjunctive arcs in G^2 corrsponding to $\Phi(S_J)$ are drawn in bold in the left graph in Figure 3-5. □

For a given S_J, $\Phi(S_J)$ can be obtained from G^J without explicitly constructing H_{G^J} as follows.

Proposition 3. *Define $\varphi(S_J) = S_J \cup \{e \in E_J : G^J(S_J \cup \overline{e})$ contains a short positive cycle passing through $\overline{e}\}$. Then $\Phi(S_J)$ is the output of the algorithm* Closure *below.*

algorithm Closure

begin

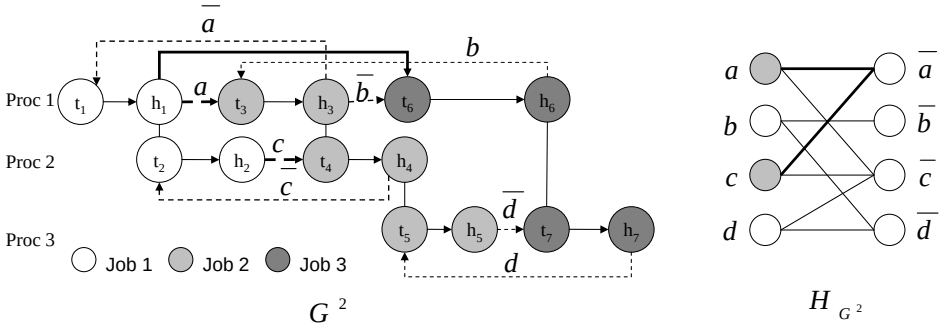

Figure 3-5: Closure in Example 3.4.

1 *Set counter* $r := 1$, $\varphi^0(S_J) := S_J$, $\Phi(S_J) := S_J$.

2 *Compute* $\varphi^1(S_J) := \varphi(S_J)$.

3 ***while*** $\varphi^r(S_J) \neq \varphi^{r-1}(S_J)$ ***do***

4 $r := r + 1$,

5 $\varphi^r(S_J) := \varphi(\varphi^{r-1}(S_J))$.

6 ***end*** *(while)*

7 $\Phi(S_J) := \varphi^r(S_J)$.

8 ***return*** $\Phi(S_J)$.

end *(*Closure*)*

Proof. Indeed, suppose $e \notin S_J$ and $e \in \varphi(S_J)$, then by the definitions of $\varphi(S_J)$, there exists some short positive cycle Z in $G^J(S_J \cup \overline{e})$ that passes through $\overline{e}$. Without loss of generality, assume that Z leaves $G^J(S_J \cup \overline{e})$ through $\overline{e}$, thus Z' must enter J through some $f \in S_J$. By construction of $H_{G^J} = (E_J, U)$, $(f, \overline{e}) \in U$, hence $f \rightsquigarrow e$ through path $(f, \overline{e}, e)$. Suppose in the next iteration, $\varphi(\varphi(S_J))$ includes $g \in E_J$, $g \notin S_J \cup e$, then there is a short positive cycle that respectively enters and leaves $G^J(S_J \cup e \cup \overline{g})$ in e and $\overline{g}$, hence $e \rightsquigarrow g$ in H_{G^J}, but then $f \rightsquigarrow g$ since $f \rightsquigarrow e$. Therefore, in iteration $r, 1 \leq r \leq |E_J|/2 - |S_J|$, when no more arcs can be added to $\varphi^{r-1}(S_J)$, we obtain the closure $\Phi(S_J)$. □

Example 3.5.

Return to the previous example with $J = 2$, $R = \{h_1, t_6\}$, and $S_J = \{(h_2, t_4)\}$. In the first iteration of the Closure algorithm, (h_1, t_3) is included into $\varphi^1(S_J)$ because of cycle $Z = (h_2, t_4, h_3, t_1, h_1, t_2)$ in G, and $\varphi^1(S_J) = \{(h_2, t_4), (h_1, t_3)\}$. After the second iteration, since $\varphi^2(S_J) = \varphi^1(S_J)$, we have $\Phi(S_J) = \{(h_2, t_4), (h_1, t_3)\}$, which is the closure found in Example 3.4. □

3.4.3 Feasible schedule generation

As pointed out in [VLAL92], when solving a classical job shop instance by the longest path problems, a new solution obtained by reversing a disjunctive arc in the critical path (i.e. replacing the arc by its mate) always produce a feasible solution. Unfortunately, such an arc exchanging scheme can no longer guarantee to obtain feasible solutions for the GBJS. A feasibility question "Given a disjunctive blocking graph G for an instance of the GBJS and an arbitrary set Q of preselected disjunctive arcs in G, is there any feasible selection S in G such that $Q \subseteq S$?" is difficult to answer. Indeed, the feasibility problem for the GBJS has been proved to be NP-complete [Kli01], which implies that applying constructive heuristics like priority-based rules to a GBJS instance might give infeasible solutions. To find a new feasible solution from a feasible one when moving an operation to another position in its processor's sequence, a new arc exchanging scheme is proposed in [GK05]. This scheme, which always finds a feasible solution, can also be derived using a main theorem on job insertion feasibility in [GK07] stated below.

Let Q be a set of preselected nodes in H_{G^J} (or equivalently, a set of preselected disjunctive arcs in G^J) and define $Q^* = \Phi(Q)$. Also define $\widehat{E}_J = \{e \in E_J : \{e, \overline{e}\} \cap Q^* = \emptyset\}$ and let $\widehat{H}_{G^J} = (\widehat{E}_J, \widehat{U})$ be the bipartite graph induced by $\widehat{E}_J$.

Theorem 4. *(i) There exists a feasible selection in G^J if and only if Q^* is stable in H_{G^J};*
(ii) If Q^* is stable in H_{G^J}, the feasible selections in G^J are precisely the sets $S_J = Q^* \cup T$, where T are the stable sets of maximum cardinality $|\widehat{E}_J|/2$ in $\widehat{H}_{G^J}$.

Proof. (i) We first prove that there exists a feasible selection in G^J only if Q^* is stable in H_{G^J}. Notice that Q must be stable in H_{G^J}, otherwise there is obviously no feasible selection in G^J that contains Q. Let S_J be a feasible selection in G^J with $Q \subseteq S_J$, then S_J is a stable set of maximum cardinality $|E_J|/2$ in H_{G^J} by Theorem 2. Suppose for $e \in Q$, $e \rightsquigarrow f$ in H_{G^J} with path $P = (e = e_0, f_1, e_1, f_2, e_2, \ldots, f_n, e_n = f)$, i.e. $f \in Q^*$. Assuming $e_{k-1} \in S_J, 1 \leq k \leq n$, as $(e_{k-1}, f_k) \in U$, we have $f_k \notin S_J$, and as $|S_J| = |E_J^-| = |E_J^+| = |E_J|/2$, $e_k = \overline{f}_k \in S_J$, thus $f \in S_J$ and $Q^* \subseteq S_J$. Therefore, S_J is feasible only if Q^* is stable in H_{G^J}. The sufficiency results from the second part of (ii) below.

(ii) Let S_J be a feasible selection in G_J with $Q \subseteq S_J$, then as proved in (i), $Q^* \subseteq S_J$. Let $T = S_J - Q^*$. By definition of $\widehat{E}_J, T = S_J \cap \widehat{E}_J$ and hence T is a stable set in $\widehat{H}_{G^J}$ since S_J is stable in H_{G^J}. As S_J has the maximum cardinality $|E_J|/2$ and $|T \cap \{e, \overline{e}\}| = 1$ for all $e \in \widehat{E}_J$, T has a maximum cardinality $|\widehat{E}_J|/2$. We next prove that if T is a stable set of maximum cardinality $|\widehat{E}_J|/2$ in $\widehat{H}_{G^J}$, then $S_J = Q^* \cup T$ is a feasible selection in G_J. Suppose that $Q^* \cup T$ is not stable in H_{G^J}, then there exists $(f, g) \in U$ in $H_{G^J} = (E_J, U)$ with $f \in Q^*$ and $g \in T$ since Q^* is stable in H_{G^J} and T is stable in $\widehat{H}_{G^J}$. As $f \in Q^*$, there is some $e \in Q$ such that $e \rightsquigarrow f$ by definition of Q^*. Then as $e \rightsquigarrow f$ and $(f, g) \in U, g \neq \overline{f}$, we have $e \rightsquigarrow \overline{g}$, hence $\overline{g} \in Q^*$, a contradiction to $\{g, \overline{g}\} \cap Q^* = \emptyset$. Therefore, $Q^* \cup T$ is stable in H_{G^J}. Since T is a stable set of maximum cardinality $|\widehat{E}_J|/2$ in $\widehat{H}_{G^J}$, $|T \cap \{e, \overline{e}\}| = 1$ for all $e \in \widehat{E}_J$, and by definitions of $\widehat{E}_J$

and Q^*, $|Q^* \cap \{e, \overline{e}\}| = 1$ for all $e \in E_J - \widehat{E}_J$. Therefore, $S_J = Q^* \cup T$ is a stable set in H_{G^J} of maximum cardinality $|E_J|/2$ and a feasible selection in G^J by Theorem 2. Observe that one of feasible selection(s) is $Q^* \cup (E_J^+ \cap \widehat{E}_J)$. □

Theorem 4 is illustrated by the following example.

Example 3.6.

Continue the previous example with $Q = \{\overline{a}\} = \{(h_3, t_1)\}$. Construct $Q^* = \Phi(Q) = \{\overline{a}, \overline{c}, \overline{d}, \overline{b}\}$, observe for instance that $\overline{a} \rightsquigarrow \overline{b}$ through $P = (\overline{a}, c, \overline{c}, d, \overline{d}, b, \overline{b})$. Q^* is stable in $H_{G^J} = H_{G^2}$ since $Q^* \subseteq E_J^+$, hence there exists a feasible selection in G^2. Construct $\widehat{E}_J = \{e \in E_J : \{e, \overline{e}\} \cap Q^* = \emptyset\}$, $\widehat{E}_J = \emptyset$ hence $T = \emptyset$. Finally, $S_J = S_2 = Q^* \cup T = \{\overline{a}, \overline{c}, \overline{d}, \overline{b}\}$ is a feasible selection in G^2. The corresponding Gantt chart is shown in Figure 3-6. Note that if $Q = \{b\}$ then $Q^* = \{b, d, c, a\}$ and $S_J = \{b, d, c, a\}$. □

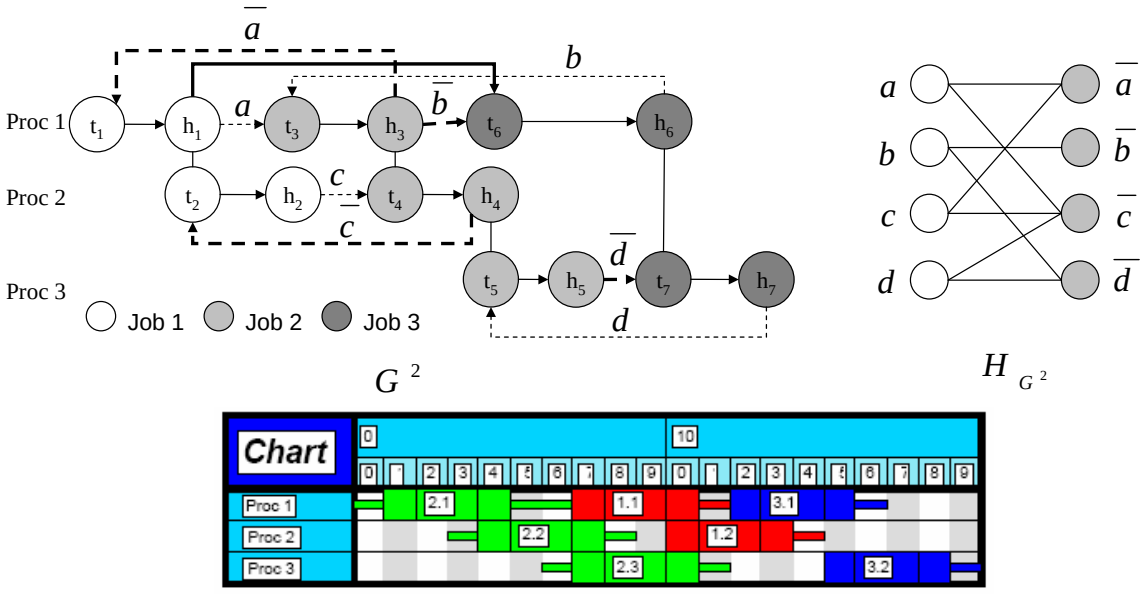

Figure 3-6: A feasible selection in G^2.

Given a disjunctive blocking graph G, a feasible selection $\overline{S}$ in G, and some arc $\overline{e} \in \overline{S}$, suppose we want to find a feasible neighbor of $\overline{S}$ in which $\overline{e}$ is replaced by its mate e. Let J be a job incident with $\overline{e}$ (w.o.l.g, always consider the head job, i.e. $head(\overline{e}) = t_i$ for some $i \in J$), and $R = \overline{S} - E_J$, construct G^J. Let $\overline{S}_J = \overline{S} \cap E_J$ and for any $K \subseteq E_J$, define $E(K) = \{e \in E : \{e, \overline{e}\} \cap K \neq \emptyset\}$. The following theorem allows us to find a new feasible insertion S_J in G^J when operation $i, t_i = head(\overline{e})$, is moved before operation $j, h_j = tail(\overline{e})$.

Theorem 5. *Let $Q = \{e = (h_i, t_j)\}$ and construct $Q^* = \Phi(Q)$. Then $S_J = Q^* \cup (\overline{S}_J - E(Q^*))$ is a feasible selection in G^J, and $S_J \cup R$ is a feasible selection in G.*

Proof. We have $e \in E_J^+$ by the selection of e, hence $Q^* = \Phi(e) \subseteq E_J^+$ by the construction of $\Phi(e)$. Therefore, Q^* is stable in H_{G^J}. Let $\overline{S}_J$ be the set of disjunctive arcs in $\overline{S}$ incident with J, then $|\overline{S}_J| = |E_J|/2$. Define $\widehat{E}_J = \{e \in E_J : \{e, \overline{e}\} \cap Q^* = \emptyset\}$ and let $\widehat{H}_{G^J} = (\widehat{E}_J, \widehat{U})$ be the bipartite graph induced by $\widehat{E}_J$. Let $T = \overline{S}_J - E(Q^*)$, then T is a stable set in $\widehat{H}_{G^J}$ since $\overline{S}_J$ is stable in H_{G^J}. Furthermore, as $|T| + |Q^*| = |\overline{S}_J|$, T is the stable set of maximum cardinality in $\widehat{H}_{G^J}$. Therefore, by Theorem 4, $Q^* \cup T$ is a feasible selection in G^J. Clearly, $S = R \cup S_J$ is complete and positive acyclic in G, hence $S = R \cup S_J$ is a feasible selection in G. □

Example 3.7.

Suppose the GBJS instance in Example 3.2 has a given initial feasible solution $\overline{S} = \{(h_1, t_6), (h_1, t_3), (h_3, t_6), (h_2, t_4), (h_7, t_5)\}$. Consider moving operation $i = 3$ before operation $j = 1$, i.e. replace $\overline{e} = (h_1, t_3)$ by $e = (h_3, t_1)$. We have $J = J^3 = 2$, hence $\overline{S}_J = \{(h_1, t_3), (h_3, t_6), (h_2, t_4), (h_7, t_5)\}$ and $R = \overline{S} - E_J = \{(h_1, t_6)\}$. For $e = (h_3, t_1)$, construct $Q^* = \Phi(e) = \{(h_3, t_1), (h_4, t_2), (h_5, t_7), (h_3, t_6)\}$ in H_{G^2} as seen in the previous example or directly from G^2 using algorithm Closure. A feasible insertion in G^J by Theorem 5 is $S_J = Q^* \cup (\overline{S}_J - E(Q^*)) = \{(h_3, t_1), (h_4, t_2), (h_5, t_7), (h_3, t_6)\}$. Finally, the feasible selection in G is $S = R \cup S_J = \{(h_1, t_6), (h_3, t_1), (h_4, t_2), (h_5, t_7), (h_3, t_6)\}$. The corresponding Gantt chart is already shown in Figure 3-6. □

After moving operation i before its current immediate processor predecessor j (i.e. replacing $\overline{e} = (h_j, t_i)$ by $e = (h_i, t_j)$), we observe in the resulting solution by Theorem 5 that (1) i is before j but not necessarily the immediate processor predecessor of j and (2) some (or all) of the operation's job J^i might also have been moved to the left, i.e. some other entering disjunctive arcs in E_J^- might have been replaced by their leaving mates in E_J^+ during the construction of Q^*. The process of moving i and possibly resequencing some other operations of J^i according to Theorem 5 is called the *closure procedure*.

3.5 Graph representation for the FGBJS

Given a feasible assignment μ where $\mu(i) \in M_i$ for all $i \in I$, a FGBJS instance becomes a GBJS one and we can associate with it a disjunctive blocking graph $G = (V, A, E, \mathcal{E}, c)$. A selection S in G only concerns sequencing operations. Consider now the move of an operation i from its current position on a processor $l = \mu(i)$ to a position on a new processor k (not necessarily different from l), i.e. the processor assignments before and after the move are μ and μ' with $\mu(i) = l, \mu'(i) = k$, and $\mu(j) = \mu'(j)$ for all $j \neq i$. Therefore, we can express the flexibility with respect to a *single* operation i by $|M_i|$ disjunctive blocking graphs, each of them associated with a distinct assignable processor $k \in M_i$. The *disjunctive blocking graph w.r.t. operation i and processor k* is denoted as $G_k^i = (V, A, E, \mathcal{E}, c)$. Clearly, given a fixed new assignment μ' (where only the processor assignment of operation i is changed), all definitions for the disjunctive blocking graph are retained for the disjunctive blocking graph w.r.t. to operation i and processor k. We have a node set $V = \{t_u, h_u : u \in I\} \cup \{\sigma, \tau\}$, a set of

conjunctive arcs $A = A^0 \cup A^1 \cup A^{\sigma,\tau}$ where $A^0 = \{(h_u, t_v), (t_v, h_u) : (u, v) \in A_J, J \in \mathcal{J}\}$, $A^1 = \{(t_u, h_u) : u \in I\}$, and $A^{\sigma,\tau} = A^\sigma \cup A^\tau = \{(\sigma, t_u) : u \in I\} \cup \{(h_u, \tau) : u \in I\}$, a set E of disjunctive arcs, $E = \{(h_i, t_j), (h_j, t_i) : i, j \in I_l, l \in M\}$, a family $\mathcal{E}$ of pairs of disjunctive arcs $D = \{e, \overline{e}\}, \cup_{D\in\mathcal{E}} = E$; and a weight function $c \in \Re^{A\cup E}$. The weight for arc $e \in A^1$ is $c_{(t_u,h_u)} = (p^t_u + p_{u,\mu'(u)})$ for $u \neq i$ and $c_{(t_i,h_i)} = \left(p^t_i + p_{ik}\right)$, $c_e = 0$ for $e \in A^0$, $c_e = p^s_{\sigma u\mu'(u)}$ for $e = (\sigma, t_u)$, $c_e = p^h_u + p^s_{\sigma u\mu'(u)}$ for $e = (h_u, \tau)$, and $c_e = p^h_u + p^s_{uv\mu'(u)}$ for $e = (h_u, t_v) \in E$. We denote the set of disjunctive arcs incident with a job J as E_J and the set of disjunctive arcs incident with operation i on processor k as E^i_k. The set of disjunctive arcs entering i is hence denoted as $E^{i-}_k = \{e \in E^i_k : head(e) = t_i\}$. Similarly, $E^{i+}_k = \{e \in E^i_k : tail(e) = h_i\}$ is the set of disjunctive arcs leaving i. For any $U \subseteq E$ in G^i_k, define $E(U) = \{e \in E : \{e, \overline{e}\} \cap U \neq \emptyset\}$.

A selection $S \subseteq E$ is *acyclic* if there is no positive cycle in $(V, A \cup S)$; S is *complete* if $S \cap D \neq \emptyset$ for all $D \in \mathcal{E}$; and S is *feasible* if it is complete and acyclic. A *solution graph* associated with operation i, processor k, and a feasible selection S in G^i_k is denoted as $G^i_k(S)$.

Given a feasible solution $(\mu, \overline{S})$ where μ is a feasible assignment and $\overline{S}$ is a feasible selection, a new graph G^i_k where operation i is moved to processor k is constructed as follows.

1. Construct the node set V, the conjunctive arc set $A = A^0 \cup A^1 \cup A^{\sigma,\tau}$, and the disjunctive arc set E with the appropriate arc weights.

2. Adjust the obtained graph if i is assigned to the same processor that performs its job predecessor and/or job successor:

   - If $k = \mu(JP(i))$ and $k \neq \mu(JS(i))$ (i.e. k is also assigned to the immediate job predecessor of i but not the immediate job successor of i): Combine i and $JP(i)$ into one operation $\widetilde{i}$ with processing time $(p_{ik} + p_{JP(i)} + p^s_{JP(i),i,k})$. In G^i_k, replace four nodes $\{t_{JP(i)}, h_{JP(i)}, t_i, h_i\}$ by two nodes $\{t_{\widetilde{i}}, h_{\widetilde{i}}\}$, replace conjunctive arcs $\{(t_{JP(i)}, h_{JP(i)}), (h_{JP(i)}, t_i), (t_i, h_{JP(i)}), (t_i, h_i)\}$ by conjunctive arc $(t_{\widetilde{i}}, h_{\widetilde{i}})$ with weight $(p^t_{JP(i)} + p_{JP(i),k} + p_{ik} + p^s_{JP(i),i,k})$, and direct disjunctive arcs entering $t_{JP(i)}$ and t_i to enter $t_{\widetilde{i}}$ and disjunctive arcs leaving $h_{JP(i)}$ and h_i to leave $h_{\widetilde{i}}$.
   - If $k \neq \mu(JP(i))$ and $k = \mu(JS(i))$: Combine i and $JS(i)$ into one operation $\widetilde{i}$ with processing time $(p_{ik} + p_{JS(i)} + p^s_{i,JS(i),k})$. In G^i_k, replace four nodes $\{t_i, h_i, t_{JS(i)}, h_{JS(i)}\}$ by two nodes $\{t_{\widetilde{i}}, h_{\widetilde{i}}\}$, replace conjunctive arcs $\{(t_i, h_i), (h_i, t_{JS(i)}), (t_{JS(i)}, h_i), (t_{JS(i)}, h_{JS(i)})\}$ by conjunctive arc $(t_{\widetilde{i}}, h_{\widetilde{i}})$ with weight $(p^t_i + p_{ik} + p_{JS(i),k} + p^s_{i,JS(i),k})$, and direct disjunctive arcs entering $t_{JS(i)}$ and t_i to enter $t_{\widetilde{i}}$ and disjunctive arcs leaving $h_{JS(i)}$ and h_i to leave $h_{\widetilde{i}}$.
   - If $k = \mu(JP(i))$ and $k = \mu(JS(i))$: Combine i, $JP(i)$, and $JS(i)$ into one operation $\widetilde{i}$ with processing time $(p_{JP(i)} + p_{ik} + p_{JS(i)} + p^s_{JP(i),i} + p^s_{i,JS(i)})$. In G^i_k, replace six nodes $\{t_{JP(i)}, h_{JP(i)}, t_i, h_i, t_{JS(i)}, h_{JS(i)}\}$ by two nodes $\{t_{\widetilde{i}}, h_{\widetilde{i}}\}$, replace conjunctive arcs $\{(t_{JP(i)}, h_{JP(i)}), (h_{JP(i)}, t_i), (t_i, h_{JP(i)}), (t_i, h_i), (h_i, t_{JS(i)}), (t_{JS(i)}, h_i), (t_{JS(i)}, h_{JS(i)})\}$

by conjunctive arc $(t_{\tilde{i}}, h_{\tilde{i}})$ with weight $(p^t_{JP(i)} + p_{JP(i),k} + p_{ik} + p_{JS(i),k} + p^s_{JP(i),i,k} + p^s_{i,JS(i),k})$, and direct disjunctive arcs entering $t_{JP(i)}, t_{JS(i)}$, and t_i to enter $t_{\tilde{i}}$ and disjunctive arcs leaving $h_{JP(i)}, h_{JS(i)}$, and h_i to leave $h_{\tilde{i}}$.

The adjustment step 2 is necessary because a moved operation can be assigned to a processor performing the operation's immediate job predecessor or immediate job successor, or both; and in these cases, there are no hand-over and take-over steps between two consecutive operations of the same job on the same processor. Construction of $G^i_k, k \in M_i$, is illustrated in the following example.

Example 3.8.

Consider a partial feasible assignment μ for the FGBJS instance in Example 3.1, $\mu(1) = 1, \mu(2) = 2, \mu(4) = 2, \mu(5) = 3, \mu(6) = 1$, and $\mu(7) = 3$. Operation 3 has as its set of alternative processors $M_3 = \{1, 3\}$. The two graphs G^3_1 and G^3_3, associated with the two choices $k = 1, k = 3$ of the processor for operation 3, are depicted respectively at the top and the bottom of Figure 3-7. Consider, for instance, G^3_3, we have

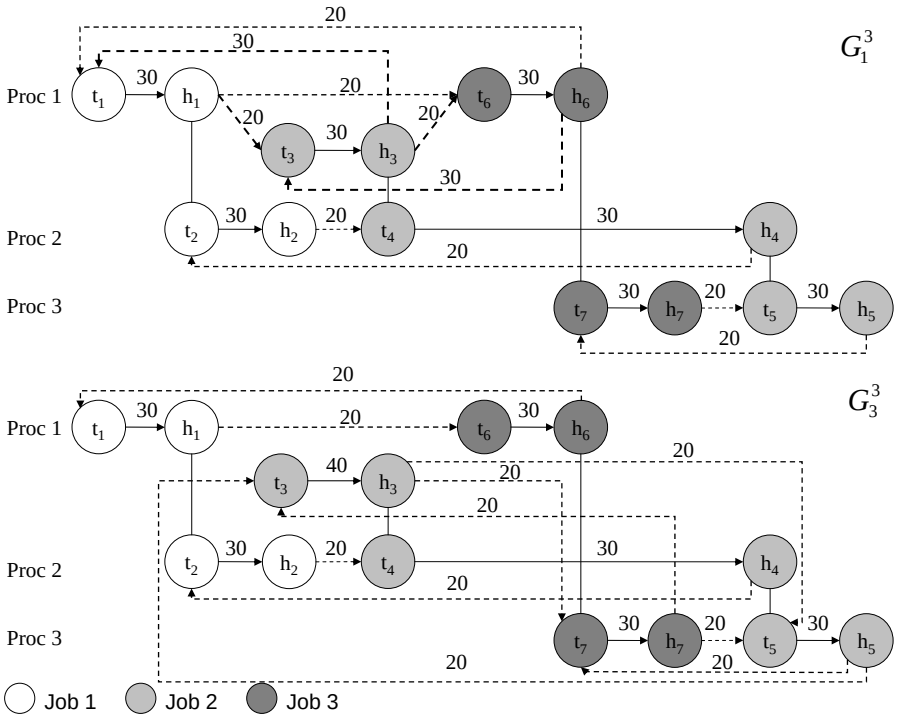

Figure 3-7: Disjunctive blocking graphs w.r.t. operation 3.

$$
\begin{aligned}
V &= \{1,\ldots,7\} \cup \{\sigma,\tau\},\\
A^0 &= \{(h_1,t_2),(t_2,h_1),(h_3,t_4),(t_4,h_3),(h_4,t_5),(t_5,h_4),(h_6,t_7),(t_7,h_6)\},\\
A^1 &= \{(t_i,h_i) : i = 1,\ldots,7\},\\
A^{\sigma,\tau} &= \{(\sigma,t_i) : i = .1,\ldots,7\} \cup \{(h_i,\tau) : i = 1,\ldots,7\},\\
E &= \{(h_3,t_5),(h_5,t_3),(h_3,t_7),(h_7,t_3),(h_1,t_6),(h_6,t_1),(h_2,t_4),(h_4,t_2),(h_7,t_5),(h_5,t_7)\},\\
\mathcal{E} &= \{\{(h_3,t_5),(h_5,t_3)\},\{(h_3,t_7),(h_7,t_3)\},\{(h_1,t_6),\ (h_6,t_1)\},\{(h_2,t_4),(h_4,t_2)\},\\
&\quad \{(h_7,t_5),(h_5,t_7)\}\}.
\end{aligned}
$$

The Gantt chart in Figure 3-8 corresponds to the following feasible selection with a preselection $\{(h_3,t_7)\}$ in G_3^3: $S = \{(h_6,t_1),(h_4,t_2),(h_7,t_5),(h_3,t_5),(h_3,t_7)\}$.

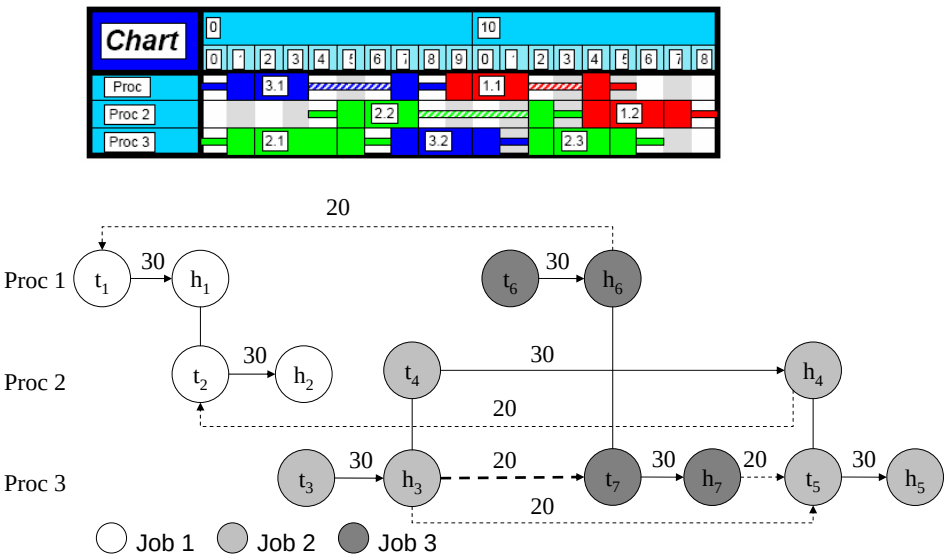

Figure 3-8: A feasible selection for Example 3.8.

If processor 2 were assignable to operation 3 with $p_{32} = 20$ and $p^s_{34} = 10$, then in addition to G_1^3 and G_3^3 we would also have G_2^3, in which operations 3 and 4 are merged into operation $\widetilde{3}$ (see Figure 3-9). □

This section ends with three remarks. First, observe that if a selection results in a zero cycle C, this implies a "*deadlock*" situation in which two or more jobs block one other (see Figure 3-10 for an example of a two-job deadlock) and all of them should be *swapped* together to reach a feasible solution. Hereinafter, swapping activities are assumed to be accepted, and such a selection having a zero cycle C is considered feasible and called *swap-based*.

The second remark is about the sequence-dependent setup times. Consider three critical operations u, v, and $w \in I$ sequenced consecutively in that order on a processor and their operational arcs lie on the critical path. If their sequence-dependent setup times do not satisfy the inequality $p^s_{uvk} + p_{vk} + p^s_{vwk} \geq p^s_{uwk}$, then as the length of the path (h_u, t_v, h_v, t_w) is not greater than the

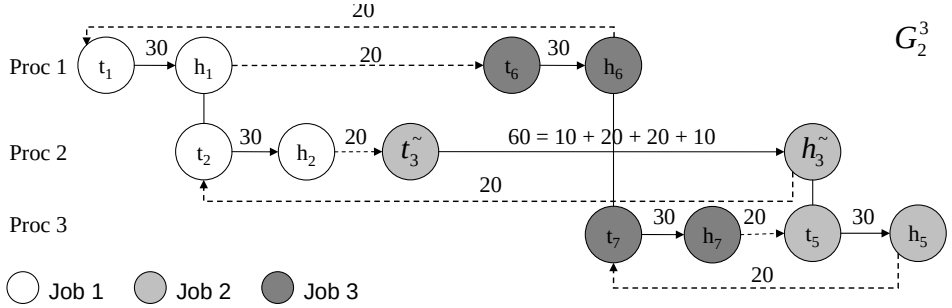

Figure 3-9: Two consecutive operations of a job are on are the same processor.

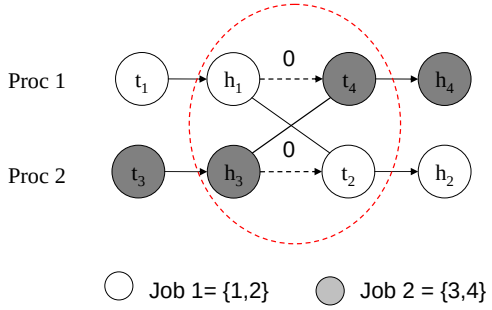

Figure 3-10: Deadlock situation involving two jobs.

length of the path (h_u, t_w), critical path computing will result in having u as the direct processor predecessor of w, which is incorrect. Therefore, the triangle inequality $p^s_{uvk} + p^s_{vwk} \geq p^s_{uwk}$ is often assumed to avoid this problem. However, in a general case, we can compute the critical path in *reduced* solution graphs where only disjunctive arcs of pairs of consecutive operations in the operation sequence of each processor are kept. Removing the transitive disjunctive arcs does not affect the operation sequence of any processor (there is a path (h_u, t_v, h_v, t_w) from h_u to t_w if disjunctive arcs (h_u, t_v) and (h_v, t_w) are kept while arc (h_u, t_w) is removed). For example, in a reduced solution graph derived from the solution graph shown in Figure 3-8, disjunctive arc (h_3, t_5) is removed.

The last remark is about solution representation. A solution for an $n \times m$ FGBJS instance can be represented as an *m-tuple* $\pi = (\pi_{1,\ldots,}\pi_m)$, where $\pi_k = (\pi_k(1), \ldots, \pi_k(l))$ denotes the operation sequence on processor k and $\pi_k(i)$ denotes the i^{th} operation performed on processor k. This representation does not need a separate list μ of the processor assigned to each operation $i \in I$ since this information can be extracted from *m-tuple* π (if $\pi_k \cap \{i\} \neq \emptyset$ then $\mu(i) = k$). Given a sequence π together with an *n-tuple* job processing order $\theta = (\theta_{1,\ldots,}\theta_{|\mathcal{J}|})$ where $\theta_J = (\theta_J(1), \ldots, \theta_J(|J|))$ is the operation sequence of job J, it is easy to determine the earliest starting time for each operation by computing the longest path $L(\sigma, i)$ for all $i \in I$ in the solution graph $G(S)$ for a selection S given by π.

3.6 Neighborhood structures for the FGBJS

Concepts of neighborhood and neighborhood structures are briefly summarized as follows. Consider the following instance of a combinatorial optimization problem ($\mathcal{P}$):

($\mathcal{P}$) Minimize $c(x)$, subject to $x \in X$,

where x is a discrete variable; $c(x)$ is some objective function; and X is a finite set of all feasible solutions.

Given a feasible solution $x \in X$, a neighbor x' of x is a feasible solution that can be reached directly from x by applying on x a transformation called *move* $s \in S(x)$, i.e. $x' = s(x), x' \in X$. Set $S(x)$ is a set of moves that can be applied to x. The *neighborhood* of x is a set of neighbors of x, i.e. $N(x) = \{x' = s(x) : x' \in X, s \in S(x)\}$ where $N : X \to 2^X$ is a *neighborhood structure* (or *neighborhood function*). N is said to be *optimum-connected* if an optimal solution x^* can be reached from each solution x, i.e. there exists a finite sequence $(x_1, x_2, \ldots, x_k)$ where $x_1 = x$, $x_{i+1} \in N(x_i)$, and $x_k = x^*$.

Neighborhood structure design has proved to substantially contribute to the performance of local search algorithms. Good neighborhood structures can result in very effective and efficient local search algorithms, such as the Tabu search by Nowicki and Smutnicki [NS96] for the JS or the Tabu search by Mastrolilli and Gambardella [MG00] for the FJS.

This section presents three neighborhood structures for the FGBJS namely *simple, pairwise exchange,* and *flexible closure neighborhood* structure. Basically, a neighbor in a neighborhood of a feasible FGBJS solution is obtained by moving an operation to another position (probably on another processor) while keeping assigned processors of other operations unchanged. In a solution obtained, other operations of the operation's job might also be moved. Denote by x be a current feasible solution for a FGBJS instance and $I^c \subseteq I$ be a set of selected operations on which moves $s \in S(x)$ are to be applied. Also denote respectively by μ and $\overline{S}$ the feasible assignment and the feasible selection for x, we write $x = (\mu, \overline{S})$. Finding a new feasible solution $x' = (\mu', S)$ from x when moving operation i to another position on processor k can be interpreted as determining a corresponding feasible selection S in disjunctive blocking graph G_k^i.

Clearly the larger the set of selected operations I^c is, the larger the resulting neighborhood becomes and the higher the chance optimality can be reached. However, a too large neighborhood is time-consuming to inspect and memory-consuming to store. For the objective function of makespan minimization, it is more computationally efficient to include into I^c only operations that will have some effect on the makespan if they are moved. Local search methods for the JS often apply moves on critical operations (operations whose nodes are on the solution graph's critical path). The rationale is that if other operations are moved then the old critical path still exists after the transformation and the makespan is not improved. We follow this approach by randomly choosing a critical path L in $G(\overline{S})$ and limiting I^c to a set of critical operations that

have at least one representing node incident with a critical arc on L (see bold lines in Figure 3-11).

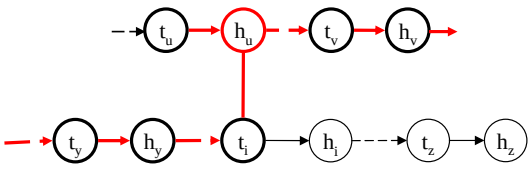

Figure 3-11: Critical path, nodes, arcs, and operations.

3.6.1 Simple neighborhood $N^s(x)$

Consider a FGBJS instance with a feasible solution x and its associated feasible assignment μ and selection $\overline{S}$. Number the q operations performed on processor $k \in M$ from 1 to q in their sequencing order and denote a position *before* operation numbered w by *position* w. Thus we have $1 \leq w \leq q+1$ where position $q+1$ denotes the position behind the last operation in the sequence.

Simple move $s \in S^s(x)$ to obtain neighbor $x' \in N^s(x)$ does the following:

1. Extract an operation i from the operation sequence of its current processor l;
2. Evaluate the feasibility when inserting i in position w in the operation sequence on processor k (not necessarily different from l), and make the insertion only if feasibility is ensured.

This move is interpreted by the following selection in $G_k^i = (V, A, E)$ (see Figure 3-12, note that $G_l^i(\overline{S})$ is the current solution graph):

1. Select all disjunctive arcs not incident with i (all $e \in E - E_k^i$) if they are selected in $\overline{S}$;
2. Select disjunctive arcs $a, b \in E_k^i, a = (h_u, t_i)$ and $b = (h_i, t_v)$ where v is the operation in position w on $k \in M_i$ and u is the immediate processor predecessor of v (if any);
3. For each operation u' and each operation v' respectively sequenced before u and after v on k, select disjunctive arcs $(h_{u'}, t_i)$ and $(h_i, t_{v'})$;
4. Accept the selection if the resulting solution graph is positive acyclic.

In step 3, when $w = 1$ only disjunctive arcs leaving i are selected and when $w = q+1$ only disjunctive arcs entering i are chosen. In the new assignment μ', only the assigned processor

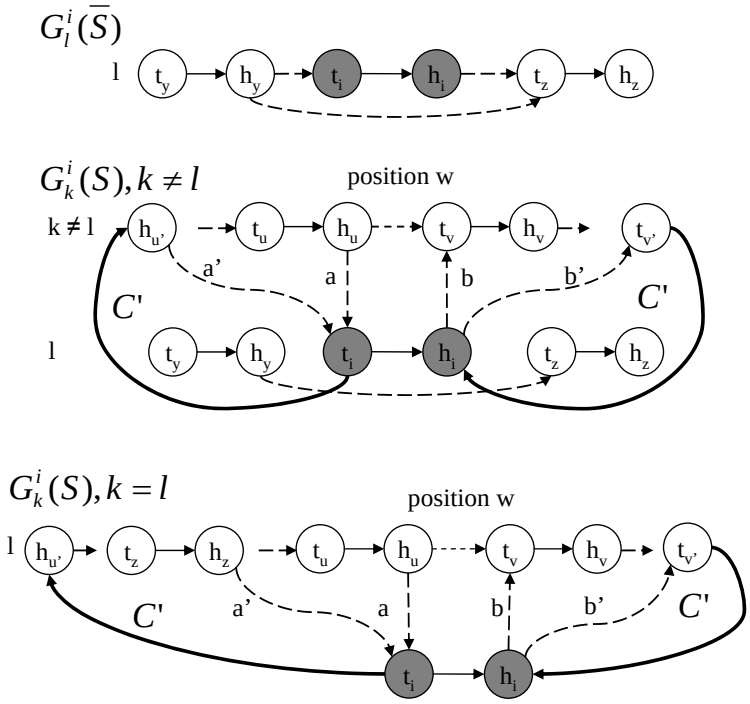

Figure 3-12: Simple move.

for operation i is possibly changed from l to k. In S, only disjunctive arcs incident with i are changed from $\overline{S}$.

A simple move is infeasible if the obtained solution graph $G_k^i(S)$ is positively cyclic. For instance, inserting i in between u and v that are involved in a swap creates infeasibility since the move results in a positive cycle consisting of a zero path P from t_v to h_u in the swap and a positive path Q with node sequence (h_u, t_i, h_i, t_v) (see Figure 3-13). Such an insertion position between u and v is called *swap-forbidden*.

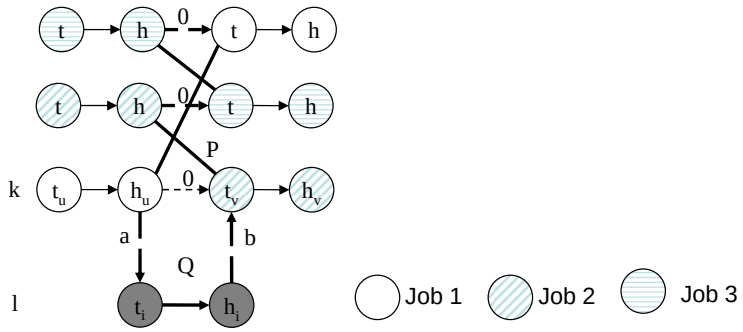

Figure 3-13: Simple move in between two operations involved in a 3-job swap.

A simple move takes $O(1)$ effort to make and $O(o)$ effort to evaluate. Suppose the largest

number of operations performed on any processor is $\overline{q} \geq 1$, then all moves $s \in S^s(x)$ associated with moving an operation i can be evaluated in $O(\overline{q}.o)$ time. However, many moves can be infeasible. Denote by E_k^{i-} and E_k^{i+} are respectively the sets of disjunctive arcs entering and leaving i in G_k^i, and let $E_k^i = E_k^{i-} \cap E_k^{i+}$. The following lemma and corollary can help us to avoid inspecting cost of some infeasible moves.

Lemma 6. *Let S be a selection resulting from moving i in between u and v on k; u and v are not involved in a swap. If a cycle C' is formed in $G_k^i(S)$, then there exists a cycle C in $G_k^i(S)$ with $C \cap E_k^i = a = (h_u, t_i)$ or $C \cap E_k^i = b = (h_i, t_v)$ (see Figure 3-12).*

Proof. Since the current selection $\overline{S}$ is feasible in G_l^i, $C' \cap E_k^i \neq \varnothing$. Suppose $C' \cap E_k^{i-} = a'$ and $C' \cap E_k^{i+} = b'$, i.e. C enters i through some $(h_{u'}, t_i)$ and leaves i through some $(h_i, t_{v'})$ (note that by the move definition, u' is before u on k when $a' \neq a$; similarly v' is after v when $b' \neq b$). Then there exists a path P from $t_{v'}$ to $h_{u'}$ which forms together with a positive path Q from $h_{u'}$ to $t_{v'}$ (as u' is before v' on k) a positive closed walk in $G_l^i(\overline{S})$, a contradiction to $\overline{S}$ being feasible in G_l^i. Therefore, either $C' \cap E_k^{i-} = a'$ and $C' \cap E_k^{i+} = \emptyset$ or $C' \cap E_k^{i-} = \emptyset$ and $C' \cap E_k^{i+} = b'$. In the former case, there is a path P from t_i to $h_{u'}$ in $G_k^i(S)$; P forms together with path Q from $h_{u'}$ to h_u and arc $a = (h_u, t_i) \in E_k^{i-}$ a cycle C in $G_k^i(S)$. C is a zero cycle only if P has length zero, $a' = a$, and a has weight zero; otherwise C is positive. In the latter case, there is a path P from $t_{v'}$ to h_i in $G_k^i(S)$, which consists of conjunctive arcs and some arcs in $\overline{S}$ not incident with i; P forms together with arc b and path Q from t_v to $t_{v'}$ a cycle C in $G_k^i(S)$. C is a zero cycle only if P has length zero, $b' = b$, and b has weight zero; otherwise C is positive. □

Corollary 7. *If there is a feasible insertion position of operation i on processor k, then the set of feasible insertion positions forms an interval $(l, l+1, \ldots, p), 1 \leq l \leq p \leq q+1$. Some positions in this interval may be swap-forbidden.*

Proof. Suppose positions l and $p, l < p$, are feasible and position $w, l < w < p$, is not feasible and not swap-forbidden (see Figure 3-14). Then there is a cycle C with $C \cap E_k^i = a = (h_u, t_i)$ or $C \cap E_k^i = b = (h_i, t_v)$ according to Lemma 6. In the first case, $C \cap E_k^i = a$ and there is a path P from t_i to h_u. Then a cycle will occur if i is inserted in any position after w; this cycle is positive since it contains at least one conjunctive arc. Therefore, position $p > w$ is not feasible, a contradiction (see the picture at the top of Figure 3-14). Similarly, in the second case where $C \cap \{a, b\} = b$, any position before w is infeasible, hence position $l < w$ is infeasible, a contradiction (see the picture at the bottom of Figure 3-14). □

Corollary 7 implies that it is unnecessary (1) to inspect further to the right if an infeasible insertion in position $p+1$ is detected and (2) to inspect further to the left if an infeasible insertion in position $l-1$ is detected. Such a feasible block $(l, l+1, \ldots, p)$ (be aware that it may contain some swap-forbidden positions) can be found by a bisection search at a computational cost of $O(\log \overline{q}.o)$. An alternative procedure is to start searching from position 0, move onward

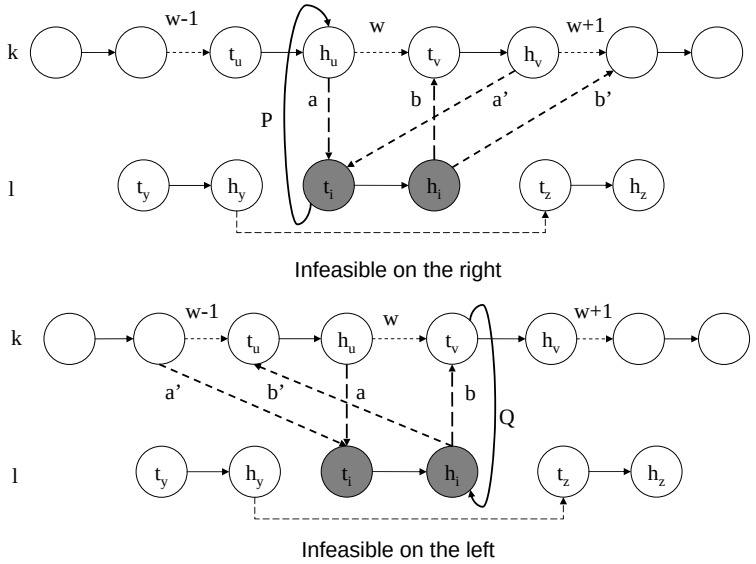

Figure 3-14: Infeasible insertions to the left and right.

to the right while recording feasible simple moves (if any), and stop at a position w behind operation u if there exists a path P from t_i to h_u since all positions after w are infeasible (see Figure 3-14). In order to save effort of explicitly constructing G_k^i and making S, these searches can be done in $G_l^i(\overline{S})$ as follows.

Proposition 8. *Let $\overline{S}_l^{i-}$ and $\overline{S}_l^{i+}$ be the set of disjunctive arcs in $\overline{S}$ that enter and leave i respectively in the current solution graph $G_l^i(\overline{S})$. A simple move will result in infeasible selection S in G_k^i if at least one of the following conditions is detected in $G_l^i(\overline{S})$: (1) There is a positive path P from t_i to h_u that does not pass through any $e \in \overline{S}_l^{i+}$; (2) $c_{(h_u,t_i)} > 0$ and there is a zero path P from t_i to h_u that does not pass through any $e \in \overline{S}_l^{i+}$;(3) There is a positive path Q from t_v to h_i that does not pass through $e \in \overline{S}_l^{i-}$; and (4) $c_{(t_v,h_i)} > 0$ and there is a zero path Q from t_v to h_i that does not pass through $e \in \overline{S}_l^{i-}$.*

Proof. Suppose there is a path P in $G_l^i(\overline{S})$ from t_i to h_u, then $P \cap \overline{S}_l^{i-} = \emptyset$ since otherwise t_i would be in a loop, contradicting $\overline{S}$ is positive acyclic. If $P \cap \overline{S}_l^{i+} = \emptyset$, then P will still be present in $G_k^i(S)$ since all arcs in $\overline{S} - \overline{S}_l^{i+}$ are reselected in S. Thus P forms with arc a a positive cycle in $G_k^i(S)$ upon two conditions (1) and (2). Similar reasoning is applied to conditions (3) and (4). □

Example 3.9.

A starting solution x for the FGBJS instance in Example 3.1 is represented by a Gantt chart at the top of Figure 3-15. The starting Gantt chart corresponds to the selection $\overline{S} = \{(h_1, t_3), (h_1, t_6), (h_3, t_6), (h_2, t_4), (h_7, t_5)\}$ in the underneath solution graph $G(\overline{S})$. Suppose we

want to move operation $i = 3$ from processor $l = 1$ to a position before operation $j = 1$ on processor $k = 1$ by a simple move. Since there is a path $P = (t_1, h_1, t_2, h_2, t_4, h_3)$ in $G(\overline{S}) = G_1^3(\overline{S})$ that does not pass through any $e \in \overline{S}_1^{3+} = \{(h_3, t_6)\}$, such a move will yield an infeasible solution. On the other hand, moving $i = 6$ from $l = 1$ to $k = 2$, before $j = 2$, is feasible since there exists no path from t_2 to h_6 in $G(\overline{S}) = G_1^6(\overline{S})$ that does not use any $e \in \overline{S}_1^{6-} = \{(h_3, t_6)\}$. The Gantt chart associated with this move is shown at the bottom of Figure 3-15.

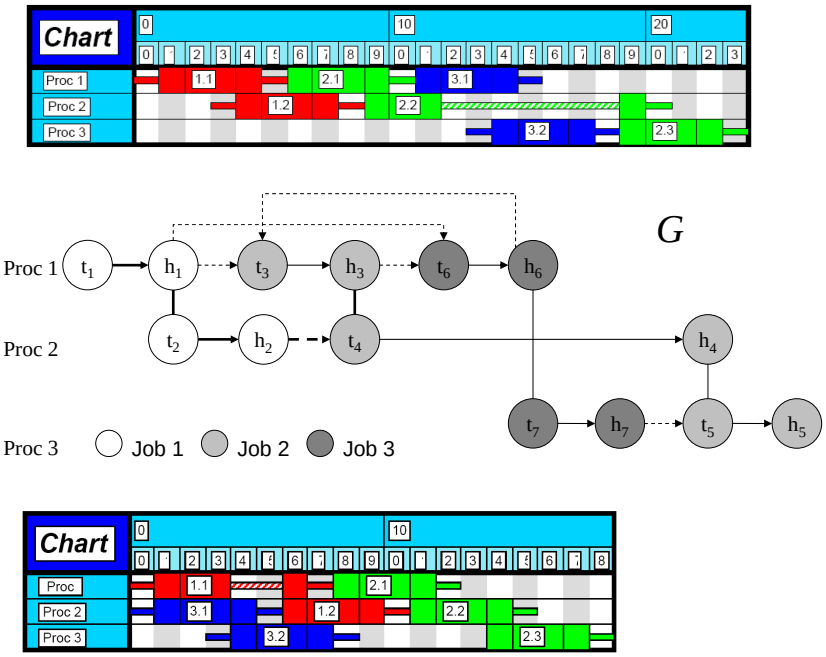

Figure 3-15: Simple neighbor for Example 3.9

Table 3-4 lists all feasible moves $s \in S^s(x) : x' = s(x), x' \in N^s(x)$, in the format of (i, u, v, k), which means that operation i is moved between operation u and v on processor k , $v = -1$ (respectively $u = -1$) indicates that i is inserted before (after) the first (last) operation $u(v)$ of the sequence, and the resulting makespan is $l_{\sigma,\tau}$.

i	u	v	k	$l_{\sigma,\tau}$	i	u	v	k	$l_{\sigma,\tau}$
6	2	-1	2	190	7	5	-1	3	240
3	7	-1	3	210	6	-1	3	1	240
7	-1	5	3	220	4	-1	2	2	240
6	1	-1	1	220	5	-1	7	3	240
6	4	2	2	220	1	3	-1	1	240
5	7	-1	3	220	3	6	1	1	240
6	3	1	1	230	2	4	-1	2	240
3	-1	6	1	230	3	5	7	3	260

Table 3-4: Simple moves in Example 3.9.

□

As we can only make a simple move when no infeasibility condition is detected, $N^s(x)$ is rather limited in space. Further, neighborhood function N^v is not optimum-connected. For instance, in the above example, after the best neighbor $(6, 2, -1, 2)$ is selected, no other improving simple move can be made while there exists a better solution of makespan 170. More complicated moves involving moving together with operation i several other operations of job J^i to recover feasibility could improve this situation.

3.6.2 Pairwise exchange neighborhood $N^p(x)$

The pairwise exchange neighborhood of a current solution x is $N^p(x) = \{x' = s(x), x' \in X, s \in S^p(x)\}$. Given a feasible solution $x = (\mu, \overline{S})$, a pairwise exchange move $s \in S^p(x)$ keeps assignment μ unchanged and generates another selection S by doing the following:

1. Exchange the position of operation i with its immediate processor predecessor j in the operation sequence of $\mu(i)$, provided that j is not a job predecessor of i;
2. Repeat resequencing several or all operations of job J^i, if necessary, while keeping i always before j (not necessarily immediately) and all currently assigned processor unchanged until feasibility is obtained.

A pairwise exchange move is precisely the construction of the selection through the closure procedure described by Theorem 5 in subsection 3.4.3. The pairwise exchange neighborhood used in [Kli01] and [GK05] consists of pairwise exchange moves associated with critical arcs. Observe that making a pairwise exchange move is more computationally expensive than making a simple move because each iteration in the Closure algorithm already involves a path detection subroutine of complexity $O(o)$ (see Appendix).

Pairwise exchange moves with the ability to recover feasibility by the closure procedure allow us to escape an impasse where no simple move exists because of highly coupling relations among operations. A pairwise exchange move brings in a more profound impact on the current solution than a simple move does since it may resequence more than one operation if necessary. However, pairwise exchange moves do not make use of processor flexibility, which is addressed in the next neighborhood structure.

3.6.3 Flexible closure neighborhood $N^c(x)$

The flexible closure neighborhood of a feasible solution x is $N^c(x) = \{x' = s(x), x' \in X, s \in S^c(x)\}$. Given a feasible solution $x = (\mu, \overline{S})$, a neighbor $x' \in N^c(x)$ in which operation i is sequenced (not necessarily immediately) *before* operation j on processor $k \in M_i$ (not necessarily different from k) is obtained by a move $s \in S^c(x)$ that does the following:

1. Extract i from its operation sequence on processor l;

2. Insert i immediately before j in the operation sequence on processor k (clearly, a move is valid only if j is not a job predecessor of i).

3. If the insertion creates infeasibility, keep resequencing the operations of job J^i if necessary until feasibility is obtained while always ensuring that (i) i precedes j, and (ii) the assigned processor for each operation of J^i is retained.

Flexible closure move s is expressed through a selection S in G_k^i as follows:

1. Select all disjunctive arcs not incident with J^i (all $e \in E - E_{J^i}$) if they are present in $\overline{S}$;

2. Select disjunctive arc $e = (h_i, t_j) \in E_k^i$, provided that $J^i \neq J^j$ or $i = J_r^i, j = J_s^i$, and $r \leq s - 1$;

3. Select disjunctive arcs in $E_{J^i} - e$ so that S is complete and $G_k^i(S)$ is acyclic.

Selecting disjunctive arcs in $E_{J^i} - e$ to obtain feasibility (in step 3 of the selection procedure above) is essentially the task of inserting job J^i into a schedule where all jobs in $\mathcal{J} - J^i$ have been scheduled so that the obtained schedule is feasible in which e is selected. To ease the notation, we denote the job of i as J in place of J^i. Note that moving i before j on k determines a new assignment μ', which differs from a current assignment μ in only the assigned processor for i. Hence, we are actually solving a GBJS instance. In G_k^i, let $R = \overline{S} \cap (E - E_J)$ and $\overline{S}_J = \overline{S} \cap E_J$. Construct the insertion graph $G^J = (V, A \cup R, E_J)$ associated with G_k^i and R, and the conflict graph $H_{G^J} = (E_J, U)$ associated with G^J. Recall that for $e, f \in E_J$, if there is an alternating path P from e to f in $H_{G^J}, P = (e = e_0, f_1, e_1, f_2, e_2, \ldots, f_n, e_n = f), n \geq 0$ where $(e_{k-1}, f_k) \in U, f_k \neq \overline{e}_{k-1}$, and $e_k = \overline{f}_k$ for all $1 \leq k \leq n$, then we write $e \rightsquigarrow f$. For any $K \subseteq E_J$, let $K^* = \Phi(K) = \{f \in E_J : e \rightsquigarrow f \text{ for some } e \in K\}$ and $E(K) = \{f \in E_J : \{f, \overline{f}\} \cap K \neq \emptyset\}$. The following theorem enables us to find a feasible job insertion in G^J when moving i before j on processor k.

Theorem 9. *Let $Q = \{e = (h_i, t_j)\}$ and construct $Q^* = \Phi(Q)$. Define $F = E_k^{i-} - E(Q^*)$ and construct $F^* = \Phi(F)$. Then $S_J = (Q^* \cup F^*) \cup (\overline{S}_J - E(Q^* \cup F^*))$ is a feasible selection in G^J, and $S_J \cup R$ is a feasible selection in G_k^i.*

Proof. By the construction of Q^*, $Q^* \subseteq E_k^{i+}$, hence Q^* is stable in H_{G^J} since H_{G^J} is a bipartite graph with two node sets $E_J = E_k^{i-} \cup E_k^{i+}$. Similarly, $F^* \subseteq E_k^{i-}$ is stable in H_{G^J}. $Q^* \cup F^*$ is a stable set in H_{G^J} because if there were $(f, g) \in U, f \in Q^*, g \in F^*$ (shown as a dashed line in Figure 3-16), then $f \rightsquigarrow \overline{g}$ and $\overline{g} \in Q^*$, contradicting $g \in F^*$. Observe that $|(Q^* \cup F^*) \cap E_k^i| = |E_k^{i-}|$ by definitions of F, Q^*, and F^*, and hence $(Q^* \cup F^*)$ defines a complete selection for i. Now consider $\widehat{E}_J = E_J - E(Q^* \cup F^*)$ and the subgraph $\widehat{H}_{G^J}$ induced by $\widehat{E}_J$. Obviously, $T = \overline{S}_J - E(Q^* \cup F^*)$ is a stable set of maximum cardinality $\left|\overline{S}_J\right| - |Q^*| - |F^*|$ in $\widehat{H}_{G^J}$. By Theorem 4, S_J is a feasible selection in G^J. Clearly, $S = R \cup S_J$ is complete and positive acyclic in G_k^i, hence $S = R \cup S_J$ is a feasible selection in G_k^i. □

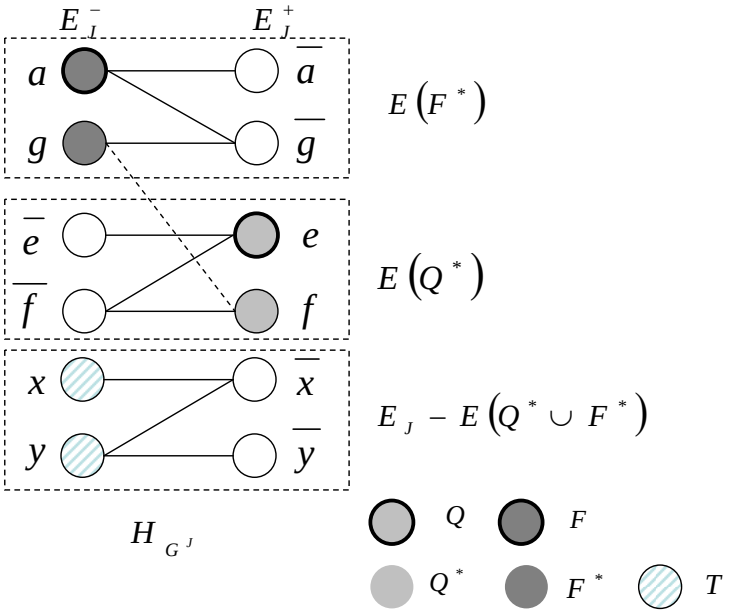

Figure 3-16: A feasible job selection in G^J or a stable set of maximum cardinality in H_{G^J}.

The process to establish a feasible selection S according to Theorem 9 is called the *flexible closure* procedure and illustrated in the example below.

Example 3.10.

The FGBJS instance in Example 3.1 is given an initial feasible solution with assignment μ : $\mu(1) = 1$, $\mu(2) = 1$, $\mu(3) = 3$, $\mu(4) = 2$, $\mu(5) = 3$, $\mu(6) = 1$, and $\mu(7) = 3$, together with a feasible selection $\overline{S} = \{(h_6, t_1), (h_2, t_4), (h_7, t_3), (h_7, t_5), (h_3, t_5)\}$. A corresponding Gantt chart is shown at the top of Figure 3-16.

Consider moving operation $i = 3$ before operation $j = 1$ on processor $k = 1$. Construct $G_k^i = G_1^3$. We have $J = J^3 = 2, \overline{S}_J = \overline{S} \cap E_J = \{(h_2, t_4), (h_7, t_5)\}$, and $R = \overline{S} \cap (E - E_J) = \{(h_6, t_1)\}$. Construct $G^J = G^2$ and let $Q = \{\overline{a} = (h_3, t_1)\}$. Find $Q^* = \{\overline{a} = (h_3, t_1), \overline{c} = (h_4, t_2)\}$, hence $E(Q^*) = \{(h_3, t_1), (h_1, t_3), (h_4, t_2), (h_2, t_4)\}$. As the set of disjunctive arcs entering i is $E_k^{i-} = E_1^{3-} = \{a = (h_1, t_3), b = (h_6, t_3)\}$, we have $F = E_k^{i-} - E(Q^*) = \{b = (h_6, t_3)\}$; hence $F^* = \{b = (h_6, t_3), d = (h_7, t_5)\}$, $Q^* \cup F^* = \{(h_3, t_1), (h_4, t_2), (h_6, t_3), (h_7, t_5)\}$ and $E(Q^* \cup F^*) = \{(h_3, t_1), (h_1, t_3), (h_4, t_2), (h_2, t_4), (h_6, t_3), (h_3, t_6), (h_7, t_5), (h_5, t_7)\}$. Therefore, $T = \overline{S}_J - E(Q^* \cup F^*) = \emptyset$ and a feasible selection in G^J according to Theorem 9 is $S_J = (Q^* \cup F^*) \cup T = \{(h_3, t_1), (h_4, t_2), (h_6, t_3), (h_7, t_5)\}$. Finally, $R \cup S_J = \{(h_6, t_1), (h_3, t_1), (h_4, t_2), (h_6, t_3), (h_7, t_5)\}$ is a feasible selection in G_1^3.

The selected arcs are drawn in bold in Figure 3-17. A corresponding Gantt chart is given at the bottom of the figure. □

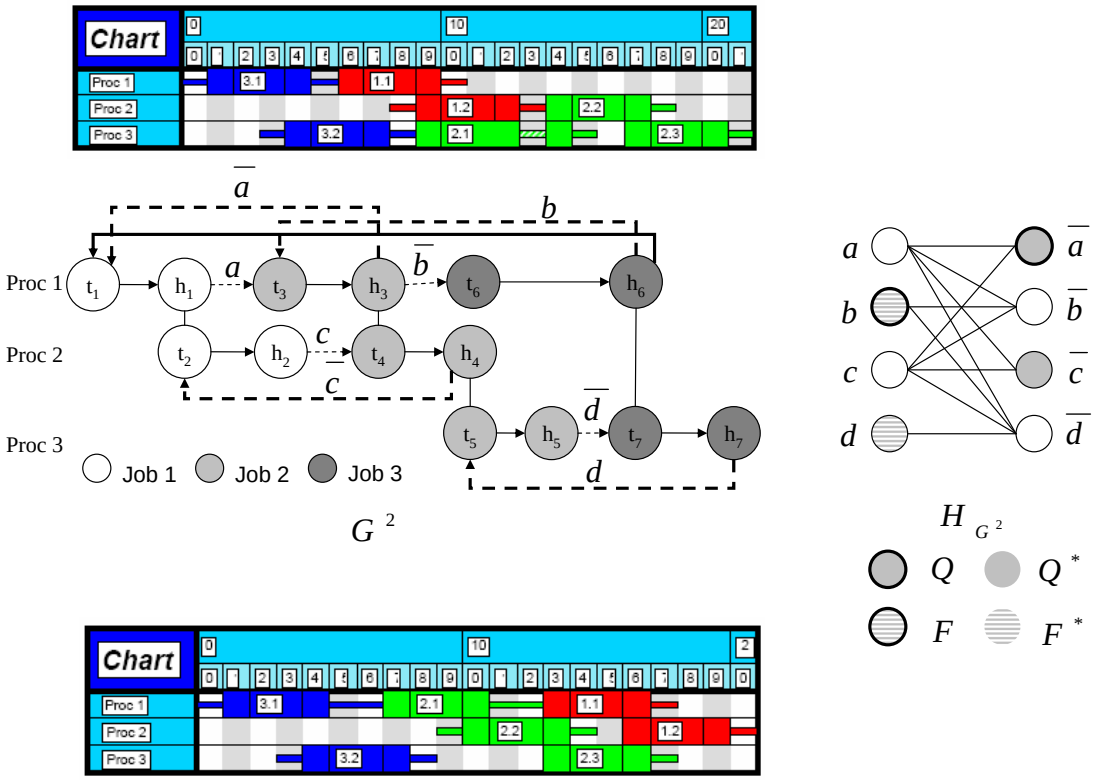

Figure 3-17: Flexible closure move of Example 3.10.

Given a FGBJS feasible solution x, its neighborhood $N^c(x)$ consists of moves associated with operations having a representative node on some critical path of x. There might be several of such critical paths. In such a case, in tracing back our critical path, we give preference to conjunctive arcs, resulting in a path containing job blocks. An example of a job block with black nodes and bold arcs is given in Figure 3-18. Moving an operation in a job block probably moves the other operations in the block; thus significant impact to the makespan may be expected.

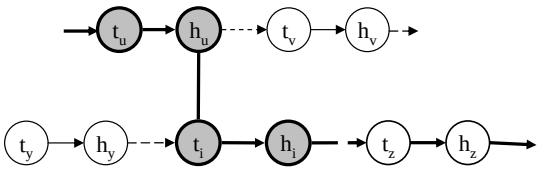

Figure 3-18: Job blocks in a critical path.

The neighborhood N^c is conjectured to be optimum-connected. Unfortunately, no proof or counter-example has been found so far.

3.6.4 Move representations

Each pairwise exchange move, as well as each flexible closure move, is represented by a pair (j, i) where i is the operation to be moved and j is the target operation that i should precede. Simple moves can also be represented by this format. However, preliminary computational experiences showed that simple moves that move an operation *behind* the last operation performed on a processor outperformed other simple moves where the operation is moved *before* another operation for many times. Therefore, we represent each simple move by a 4-tuple (i, u, v, k) where i denotes the operation to be moved, k the new processor for i, and u and v two consecutive operations on k to insert i in between. When v is the first operation in the sequence on k, set $u := -1$. Similarly, if u is the last operation in the sequence then $v := -1$.

3.7 Constructive heuristics

This section presents several heuristics to construct feasible solutions. Obviously, for a given feasible assignment, any randomly generated permutation of jobs yields a feasible solution, a so-called permutation schedule. Nevertheless, non-trivial constructive heuristics can give solutions of much better quality.

3.7.1 A general framework for constructive heuristics

The three constructive heuristics to be introduced in this section share a base algorithm called *Schedule Generator (SG)*. Basically, the SG schedules all jobs in a given job sequence one by one. For each job, it solves an embedded job insertion problem. Recall that in a job insertion problem, given a partial schedule of some jobs and a newly arrived job, we need to insert the new job into the partial schedule so that a resulting makespan is minimized. The job insertion problem for the JS has been proved to be NP-hard [KH03]; thus the job insertion problem for the FGBJS is also NP-hard, hence heuristics are often used to solve the problem.

The SG consists of two parts. Part 1 (lines 1 to 11) determines the insertion order for the jobs based on their average processing times and schedule the first ordered job. The average processing time of an operation i is computed as $\overline{p}_i = \left(\sum_{k \in M_i} p_{ik}\right) / |M_i|$ and the average processing time of job J is $\overline{P}_J = \sum_{i \in J} \overline{p}_i$. The initial fixed workload of processor k is $W_k^0 = \sum_{i \in I: M_i = \{k\}} (p_i^t + \overline{p}_i + p_i^h)$. After ordering jobs to insert and computing the fixed initial workload for each processor, the algorithm assigns each operation of the first job in the order to a valid processor having the smallest initial fixed workload. In the case of tie, assign the operation to the processor of the smallest index. After the first job in the order is scheduled, in part 2, the heuristic consecutively schedules the other jobs according to a particular insertion algorithm A (line 14).

algorithm *(SG)*

begin

1 Order all n jobs in the non-increasing order of total average processing times.

2 Label the jobs from 1 to n according to the order above.

3 Calculate the initial fixed workload W_l^0 for every processor $l \in M$.

4 **for each** operation i of the job labeled 1 **do**

5 **if** $M_i = \{k\}$ **then** set $\mu(i) := k$ and $p_i := p_{ik}$.

6 **else**

7 Set $\mu(i) := l$ where $l \in M_i$ has the least fixed workload: $W_l^0 \leq W_{l'}^0$ for all $l' \in M_i$

8 Set $p_i := p_{il}$.

9 Update the processor's fixed workload $W_l := W_l^0 + p_{il} + p_i^t + p_i^h + p_{\sigma i}^s$.

10 **end** (else)

11 **end** (for)

12 Sequence each operations of J as the first operation on its assigned processor.

13 **for each** job labeled from 2 to n **do**

14 Call job insertion algorithm A.

end *(SG)*

The following example shows how to order the jobs and schedule the first job.

Example 3.11.

We illustrate part 1 of the SG algorithm through the FGBJS instance in Example 3.1. To facilitate reading, the data in Example 3.1 are recalled as follows. $\mathcal{J} = \{1, 2, 3\}$, $M = \{1, 2, 3\}$, $I = \{1, 2, ..., 7\}$, operations of job 1 is $\{1, 2\}$, job 2 is $\{3, 4, 5\}$, and job 3 is $\{6, 7\}$. Sets of assignable processors for operations are $M_1 = \{1\}$, $M_2 = \{2\}, M_3 = \{1, 2, 3\}, M_4 = \{2\}$, $M_5 = \{3\}, M_6 = \{1, 2, 3\}$, and $M_7 = \{3\}$. All take-over and hand-over times, as well as the first and last setup times, are equal to 10. Processor-dependent processing times and the sequence-dependent setup times are respectively shown below; all setup times are processor-independent.

Proc/Op	1	2	3	4	5	6	7
1	20		20			20	
2		20	20	20		20	
3			30		20	30	20

Op/Op	1	2	3	4	5	6	7
1			10			10	
2				10		10	
3	20				10	10	10
4		10				10	
5							10
6	10	10	20	10	10		10
7			10		10		

Average processing times for the operations are $\overline{p}_1 = 20$, $\overline{p}_2 = 20$, $\overline{p}_3 = (20+20+30)/2 = 23.33$, $\overline{p}_4 = 20$, $\overline{p}_5 = 20$, $\overline{p}_6 = (20+20+30)/2 = 23.33$, and $\overline{p}_7 = 20$. Average processing times for the jobs are $\overline{P}_1 = 40$, $\overline{P}_2 = 63.33$ and $\overline{P}_3 = 43.33$; hence the jobs are ordered as $\{2, 3, 1\}$. Processor fixed workloads are $W_1^0 = p_1^t + p_{11} + p_1^h = 40$, $W_2^0 = p_2^t + p_{22} + p_2^h + p_4^t + p_{42} + p_4^h = 80$, and $W_3^0 = p_5^t + p_{53} + p_5^h + p_7^t + p_{73} + p_7^h = 80$. The operations of job 2 are assigned to processors as follows:

1. Assignable processors for operation 3 are $M_3 = \{1, 2, 3\}$, $\min(W_1^0, W_2^0, W_3^0) = 40$, so $\mu(3) := 1$. Update the fixed workload of processor 1, $W_1 := W_1^0 + p_{31} + p_3^t + p_3^h + p_{\sigma 31}^s = 90$.
2. Operation 4 has $M_4 = \{2\}$, hence $\mu(4) := 2$.
3. Operation 5 has $M_5 = \{3\}$, hence $\mu(5) := 3$.

Each operation of job 1 is now the first operation on its assigned processor. □

A constructive heuristic developed from the SG is named after the job insertion subroutine heuristic A it uses (see line 14) as *SG-A*. There are three such job insertion heuristics namely *Most Critical Operation (MOC), Most Favorable Position (MFP),* and *Bottleneck (BN).* So there are three constructive heuristics called *SG-MOC, SG-MFP,* and *SG-BN.* In general, each job insertion heuristic inserts a new job into a partial schedule in three steps:

1. Assign each operation of the job to one of its assignable processors that has the lightest fixed load;
2. Append the job's operations to the assigned processor's operation sequence;
3. Improve the resulting schedule.

Three heuristics MOC, MFP, and BN differ only in the last step of improving a schedule obtained after appending the new job to an existing schedule.

3.7.2 Most critical operation heuristic

The job insertion heuristic Most Critical Operation (MCO) is based on an idea that each job typically has an operation that is the most "expensive" in terms of processing time or cost,

while the other job's operations play supporting roles. For instance, the most critical operation of a surgery is its perioperative operation. Scheduling the critical operation should determine scheduling for other operations.

algorithm *(MCO)*

begin

for each operation i of job J **do**

if $M_i = \{k\}$ **then** set $\mu(i) := k$ and $p_i := p_{ik}$.

else

for each $k \in M_i$

Calculate the potential workload $\overline{W}_k := W_k + p^s_{qik} + p^t_i + p_{il} + p^h_i$,

where q is the last operation on k in the current partial schedule.

end (for)

Set $\mu(i) := l$, $l \in M_i$ has the least potential workload: $\overline{W}_l \leq \overline{W}_k \ \forall k \in M_i$.

Set $p_i := p_{il}$.

Update the processor's fixed workload, $W_l := W_l + p^s_{qil} + p^t_i + p_{il} + p^h_i$,

where q is the last operation on l in the current partial schedule.

end (else)

end (for)

Append each $i \in J$ to the sequence on $\mu(i)$ to obtain a feasible solution x.

Calculate the partial makespan $c(x)$ and set $c^* := c(x)$.

Find the most critical operation u of J.

Generate a list of flexible closure moves $S^c(x)$,

$S^c(x) = \{(j, u) : \mu(j) = \mu(u), j$ is sequenced before u and behind any job

predecessor of u performed on the same processor in the current partial schedule$\}$.

Set the best candidate move $s^* := \emptyset$.

for $k = 1$ to $S^c(x)$ **do**

if $c(s_k(x)) < c^*$ **then** $s^* := s_k(x)$, $c^* := c(s_k(x))$ where $s_k \in S^c(x)$.

end (for)

if $s^* \neq \emptyset$ then obtain a new solution $x' := s^*(x)$.

end *(MCO)*

After a processor for each operation of J has been determined, append J to the partial schedule

to obtain a new schedule. Then determine the most critical operation u of the job J by ranking the job's operations in a non-decreasing order of their average processing times. Starting from the first operation in this order, choose the first operation in the list that has some predecessor on its assigned processor as the critical operation of J. Next, improve the initial schedule where J is appended by evaluating moves associated with moving the critical operation u to different positions on its assigned processor, then making the best move if it improves the initial schedule.

Example 3.12.

Algorithm SG-MOC determines a job sequence $\{2, 3, 1\}$ and the processor for each operation of job 2 as previously determined in Example 3.11. Consider inserting job 3:

1. Operation 6 has $M_6 = \{1, 2, 3\}$, the potential workloads $\overline{W}_1 = 90 + p_{61} + p_6^t + p_6^h + p_{36}^s = 140$, $\overline{W}_2 = 80 + p_{62} + p_6^t + p_6^h + p_{46}^s = 130$, and $\overline{W}_3 = 80 + p_{63} + p_6^t + p_6^h + p_{56}^s = 140$, hence $\mu(6) := 2$. Update $W_2 := \overline{W}_2 = 130$.

2. Operation 7 has $M_7 = \{3\}$, hence $\mu(7) := 3$.

Operations 6 and 7 are appended to the current partial schedule, resulting in a new partial schedule of makespan 170. As the average processing time of operation 6 is larger then the one of operation 7, $\overline{p}_6 > \overline{p}_7$, the critical operation of job 3 is 6. There is only one position for operation 6 to move in the operation sequence on processor 2, that is before operation 4. This is an improving move of makespan 140; so the move is made. Continue with the last job (job 1), we have $\mu(1) := 1$ and $\mu(2) := 2$. Appending operations 1 and 2 to the current partial schedule (behind operation 6 and 4 respectively) results in a schedule of makespan 170. The critical operation of job 1 is operation 1 and its single associated move $(3, 1)$ would lead to a larger makespan of 190, therefore operations of job 1 stay in their current positions. The Gantt chart of the final schedule is shown in Figure 3-19.

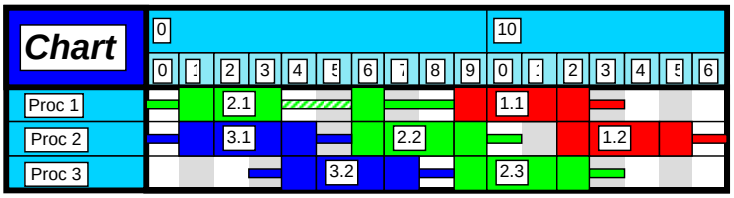

Figure 3-19: Feasible schedule obtained by the SG-MOC for Example 3.12

3.7.3 Most favorable position heuristic

The job insertion heuristic Most Favorable Position (MFP) considers all rather than just a single "critical" operation of the inserted job in the improving step (step 3). It finds the

best improving position for each operation by estimating the decrease in the time span of the operation's processor for each position instead of calculating an exact makespan associated with each position for each operation. Suppose there are q operations sequenced on processor k and labeled $(1, \ldots, q)$ in that sequence. Let λ_k be the departure time of the last operation on k, $\lambda_k = x_q^t + p_q^t + p_{qk} + p_q^h$, and refer to λ_k as the *time span* of k. Consider moving the last operation of the sequence, operation labeled q, before operation i, $q-1 \geq i \geq 1$:

1. Moving operation q before operation labeled i and after operation labeled $i-1, q-1 \geq i \geq 2$, *decreases* λ_k by an amount $\triangle_i^k = (x_i^t - \left(x_{i-1}^h + p_{i-1}^h\right)) - (p_{i-1,q,k}^s + p_{qik}^s) + p_{q-1,q,k}^s$, where $(x_i^t - \left(x_{i-1}^h + p_{i-1}^h\right))$ is the interval between the departure of operation $i-1$ and the start of operation i;

2. Moving operation q before operation $i = 1$ decreases λ_k by an amount $\triangle_1^k = x_1^t - p_{q,1,k}^s + p_{q-1,q,k}^s$.

Positions yielding $\triangle_i^k > 0, 1 \leq i \leq q-1$, are favorable to those yielding negative values because they decrease the time span of k. The position yielding the largest decrease in time span is called the *most favorable position*, and the operation corresponding to this position is denoted by $\triangle(q)$. If there is no favorable position for q then $\triangle(q) := \emptyset$.

Details of the MFP heuristic are found below.

algorithm *(MFP)*

begin

for each operation i of job J **do**
  if $M_i = \{k\}$ **then** set $\mu(i) := k$ and $p_i := p_{ik}$.
  else
    for each $k \in M_i$
      Calculate the potential workload $\overline{W}_k := W_k + p_{qik}^s + p_i^t + p_{il} + p_i^h$,
      where q is the last operation on k in the current partial schedule.
    end (for)
    Set $\mu(i) := l$, $l \in M_i$ has the least potential workload: $\overline{W}_l \leq \overline{W}_k \ \forall k \in M_i$.
    Set $p_i := p_{il}$.
    Update the processor's workload, $W_l := W_l + p_{qil}^s + p_i^t + p_{il} + p_i^h$,
    where q is the last operation on l in the current partial schedule.
  end (else)
end (for)

Append each $i \in J$ to the sequence on $\mu(i)$ to obtain a feasible solution x.

Calculate the partial makespan $c(x)$.

Set the list of flexible closure moves $S^c(x) := \emptyset$.

for each operation i of J **do**

Find the most favorable position for i and its associated target operation $\triangle(i)$.

Set $S^c(x) := S^c(x) \cup (\triangle(i), i)$.

end (for)

Set the best candidate move $s^* := \emptyset$ and $c^* := c(x)$.

for $k = 1$ to $S^c(x)$ **do**

if $c(s_k(x)) < c^*$ **then** $s^* := s_k(x)$, $c^* := c(s_k(x))$ where $s_k \in S^c(x)$.

end (for)

if $s^* \neq \emptyset$ then obtain a new solution $x' := s^*(x)$.

end (*MFP*)

Example 3.13.

Continue Example 3.11 with a determined job insertion sequence $\{2, 3, 1\}$. Further assume that jobs 2 and 3 have already been scheduled and sequences on the processors are $\pi_1 = \{3\}, \pi_2 = \{6, 4\}$, and $\pi_3 = \{7, 5\}$. Apply the MFP algorithm to schedule job 1 as follows. Assign each operation of job 1 to its single processor ($\mu(1) := 1$ and $\mu(2) := 2$) and append job 1 to the current partial schedule to obtain a schedule of makespan 170. For operation 1, its single possible position before operation 3 would give a decrease of the time span on processor 1 of $\triangle_3^1 = x_3^t + p_{3,1,1}^s - p_{1,3,1}^s = 20$. Thus, this position is favorable and $\triangle(1) := 3$. Calculations for operation 2 with respect to operations 6 and 4 on processor 2 give $\triangle_4^2 = (x_4^t - x_3^h - p_3^h) - (p_{3,2,2}^s + p_{2,6,2}^s) + p_{4,2,2}^s = 0$ and $\triangle_6^2 = x_6^t + p_{6,1,2}^s - p_{1,6,2}^s = 10$, hence $\triangle(2) := 6$. As flexible closure moves $(3, 1)$ and $(6, 2)$ are not improving, no further move is made. The Gantt chart of the final schedule is shown in Figure 3-20.

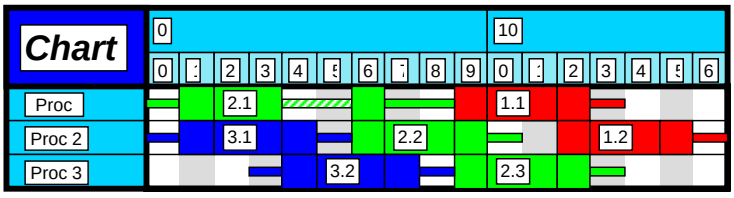

Figure 3-20: Feasible schedule obtained by the SG-MFP for Example 3.13.

3.7.4 Bottleneck heuristic

In the improving step of the job insertion heuristic Bottleneck (BN), operations performed by the most loaded processor, the so-called bottleneck, are resequenced. Resequencing is done by making the best pairwise exchange move among all moves associated with the operations on the bottleneck. Details of the heuristic are found below.

algorithm *(BN)*

begin

for each operation i of job J **do**

if $M_i = \{k\}$ **then** set $\mu(i) := k$ and $p_i := p_{ik}$.

else

for each $k \in M_i$

Calculate the potential workload $\overline{W}_k := W_k + p^s_{qik} + p^t_i + p_{il} + p^h_i$,

where q is the last operation on k in the current (partial) schedule.

end (for)

Set $\mu(i) := l$, $l \in M_i$ has the least potential workload: $\overline{W}_l \leq \overline{W}_k \ \forall k \in M_i$.

Set $p_i := p_{il}$.

Update the processor's workload, $W_l := W_l + p^s_{qil} + p^t_i + p_{il} + p^h_i$,

where q is the last operation on l in the current partial schedule.

end (else)

end (for)

Append each $i \in J$ to the sequence on $\mu(i)$ to obtain a feasible solution x.

Calculate the partial makespan $c(x)$.

for each processor $k \in M_J$ (i.e. k is assigned to an operation of J) **do**

Label q operations already on k from 1 to q in their sequence order.

Calculate the actual load on k:

$\widehat{W}_k = p^s_{\sigma 1k} + \sum_{i=2}^{q-1} p^s_{i,i+1,k} + p^s_{q\tau k} + \sum_{i=1}^{q} (p^t_i + p_{ik} + p^h_i)$.

end (for)

Find the bottleneck $l \in M_J : \widehat{W}_l \leq \widehat{W}_k$ for all $k \in M_J$

Generate a list of pairwise moves

$S^p(x) = \{(j,i) : \mu(i) = \mu(j) = l, j$ immediately precedes i on the found bottleneck $l\}$

Set best candidate move $s^* := \emptyset$. Set $c^* := c(x)$.

25 **for** $k = 1$ to $|S^p(x)|$ **do**

26 **if** $c(s_k(x)) < c^*$ **then** $s^* := s_k(x)$, $c^* := c(s_k(x))$, where $s_k \in S^c(x)$.

27 **end** (for)

28 **if** $s^* \neq \emptyset$ then obtain a new solution $x' := s^*(x)$.

end *(BN)*

Example 3.14.

Continue Example 3.11 with the job sequence $\{2, 3, 1\}$ and job 2 has been scheduled. Schedule job $3 = \{6, 7\}$. Assume that the processor assignment step for job 3 has been done, which gives $\mu(6) := 2$ and $\mu(7) := 3$. Appending job 3 to the current partial schedule yields a makespan 170. Calculate the actual workloads for processors in $M_{J=3} = \{2, 3\}$ as $\widehat{W}_2 = p^s_{\sigma 42} + p^t_4 + p_{4,2} + p^h_4 + p^s_{4,6,2} + p^t_6 + p_{6,2} + p^h_6 = 110$ and similarly, $\widehat{W}_3 = 110$. The tie-breaking rule selects processor 2 as the bottleneck whose list of pairwise exchange moves is $\{(4, 6)\}$, which is an improving move with the corresponding makespan of 140. Next, continue with the last job (job 1) with $\mu(1) := 1$ and $\mu(2) := 2$. After job 1 is appended, calculate the workloads of processors 1 and 2 to find that the bottleneck is processor 2 with $\widehat{W}_2 = 110$. Since none of the bottleneck's pairwise exchange moves $\{(6, 4), (4, 2)\}$ is an improving one, we keep the current schedule with makespan 170 (see Figure 3-21).

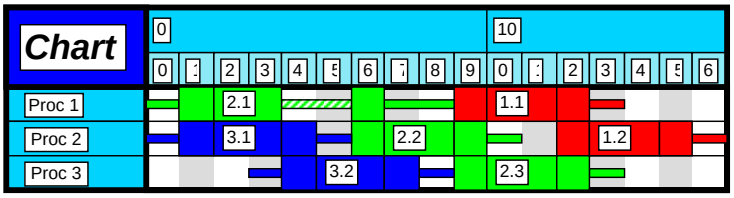

Figure 3-21: Feasible schedule obtained by the SG-BN for Example 3.14.

3.8 Tabu search algorithms

Tabu search (TS) [GL97] is frequently reported in in scheduling literature as a successful tool for solving job shop related problems (e.g. in [VAL96][MG00]). For this reason, we selected TS as our main solution method for the FGBJS. In this section, three generic TS algorithms are presented. Next, these generic algorithms are combined with the three different neighborhood structures discussed in Section 3.6 to create six TS algorithms for the FGBJS.

3.8.1 Three generic Tabu search algorithms

Tabu search is a deterministic search algorithm, which is used as a general tool to solve combinatorial optimization problems [GL97]. Given a combinatorial optimization problem ($\mathcal{P}$), TS starts with a feasible initial solution and solves ($\mathcal{P}$) iteratively by selecting a move in a set of moves applicable to the current solution and applying the move to the solution to obtain a new solution. Making a move made in some earlier iteration might lead to a previously found solution. To prevent such a solution cycling, all found solutions should be recorded. However, keeping a record of all found solutions is memory-consuming while checking the record is time-consuming. That is why only some recent solutions are stored. To save memory and inspection time, these solutions are not stored directly but indirectly through so-called *tabu* moves that have created the solutions. These tabu moves are stored in a list T of size t called *tabu list*. When a move s is applied to a solution x to find its neighbor x', the reverse move s^{-1}transforming x' back to x is appended to T, and if T is already full then the oldest move in T is dropped from T. This is to ensure that we will not get back to solution x for at least t iterations. Note that since only some attributes of a solution are stored, a tabu move may correspond to some unvisited solutions. The TS algorithm stops when a stopping criterion is met (e.g. the maximum number of iterations ($\max iter$) is reached; or the maximum number of non-improving iterations is passed; or the allotted running time is over, etc).

The initial TS algorithm (TSB) proposed by Glover is presented below:

algorithm *TSB*

begin

1 Calculate an initial feasible solution x.

2 Set the best solution so far $x^* := x$ and the best objection function value $c^* = c(x^*)$.

3 Set tabu list $T := \emptyset$ and the iteration counter $k := 0$.

4 **while** $k <$ max *iter* and other stop criteria (if any) are *not* met **do**

5 Set $k := k + 1$.

6 Find the candidate list of moves $S(x)$.

7 **if** $S(x) - T = \emptyset$ **then stop**.

8 **else** select the best move $s_k \in S(x) - T$.

9 Set $x := s_k(x)$.

10 Update T with the reverse move s_k^{-1}.

11 **if** $c(x) < c^*$ **then** set $x^* := x$, $c^* := c(x)$.

12 **end** (while)

end (*TSB*)

Note that in iteration k, the best move to be selected is typically the move that gives the best objective function value, i.e. $c(s_k(x)) \leq c(s'_k(x) : s' \in S(x) - T)$.

The TSB has been enhanced by various advanced techniques, e.g. applying aspiration criteria, varying the tabu list length, deploying long-term memory strategies to improve the search quality, etc. One of such improvised TS versions is *Tabu Search with Back Jump Tracking (TSBJT)* algorithm developed by Nowicki and Smutnicki to solve the JS [NS96]. TSBJT uses two advanced techniques: *aspiration* criteria and *elite list*. The first technique is used to select a move when all possible moves are tabu. Two aspiration criteria in [NS96] are: (1) *aspiration by default* where the oldest tabu move is selected, and (2) *aspiration by improvement* where a tabu move improving the current best objective function is chosen. The second technique employs an *elite list* $\mathcal{L}$ of fixed size l as a long-term memory instrument to both intensify and diversify the search. If a solution x_k is an improving one, it is set to be appended to $\mathcal{L}$. Suppose in iteration $k+1$, a predefined number of non-improving moves is reached or objective function value's cycling is detected, the search thus comes back to $\mathcal{L}$ to use the last stored improving solution as a new starting solution to *intensify* the search in a seemingly good region. Restarting the search from scratch looses all information found so far. Therefore besides x_k, $\mathcal{L}$ also stores the tabu list T_k found in iteration k and the set of applicable moves $S(x_k)$ found in iteration $k+1$. As a result, $\mathcal{L}$ is a list of triples, $\mathcal{L} = \{\mathcal{L}_1, \ldots, \mathcal{L}_e\}$ where $\mathcal{L}_r = (x_k, T_k, S(x_k))$, $1 \leq r \leq e \leq l, k \geq r$. When $\mathcal{L}$ is full and a new triple is to be added to $\mathcal{L}$, the oldest one is dropped from $\mathcal{L}$. When the last improving solution x_k in $\mathcal{L}$ is called, the previously selected move s_k applied to x_k should not be reapplied to avoid repeating the previously found search path. This search *diversification* is achieved as follows. In iteration k, x_k and T_k are not yet stored in $\mathcal{L}$. In the next iteration $k+1$, before applying the selected move s_{k+1} to obtain the new solution $x^{k+1} = s_{k+1}(x_k)$, a triple $\mathcal{L}_k = (x_k, T_k, S(x_k) - s_{k+1})$ is appended to $\mathcal{L}$.

The pseudo-code of the TSBJT is given below, in which new parameter $\max q < maxiter$ is the maximum number of consecutive non-improving moves.

algorithm *TSBJT*

begin

1 Calculate an initial feasible solution x.

2 Set the best solution so far $x^* := x$ and the best objection function value $c^* = c(x^*)$.

3 Set tabu list $T := \emptyset$ and the iteration counter $k := 0$.

4 Set the flag $save := true$, the flag $restore := false$, and the elite list $\mathcal{L} := \emptyset$.

5 **while** $k <$ max $iter$ and the allotted running time is not over **do**

6 Set $k := k + 1$.

7 **if** $restore = false$ **then** find the candidate list of moves $S(x)$.

8 **else** set $restore := false$.

9 **if** $S(x) - T = \emptyset$ **then**

10 Apply aspiration by improvement to choose the best improving tabu move

11 $s_k \in S(x)$. If no improving move exists then apply aspiration by default to choose

12 the oldest move $s_k \in T \cap S(x)$.

13 **else** choose the best move $s_k \in S(x) - T$.

14 **if** $save = true$ and $|S(x)| > 1$ **then** update $\mathcal{L}$ with the triple $(x, T, S(x) - s_k)$.

15 Set $save := false$;

16 Update T with the reverse move s_k^{-1}.

17 Set $x := s_k(x)$.

18 **if** $c(x) < c^*$ **then**

19 Set $x^* := x$, $c^* := c(x)$, $save := true$, and **continue** with the next iteration.

20 **if** $k < \max q$ and no cycling is detected on $c(x)$ **then continue** with the next iteration.

21 **if** $\mathcal{L} = \emptyset$ then **stop**.

22 Given $\mathcal{L} = \{\mathcal{L}_1, \ldots, \mathcal{L}_{e:e \leq l}\}$, set $x := x_e, T := T_e, S(x) := S_e, \mathcal{L} := \mathcal{L} - \mathcal{L}_e$.

23 Set $save := true$, $restore := true$, $k := 0$. Update the running time so far.

24 **end** (while)

end (*TSBJT*)

The aspiration by default gets the oldest move in $T \cap S(x)$ by left-shifting T, i.e. $T_i := T_{i+1}$ $(i = 1, \ldots, t-1)$ until $S(x) - T \neq \emptyset$. The elite list $\mathcal{L} = \{\mathcal{L}_1, \ldots, \mathcal{L}_e\}$ is updated with a new triple $\mathcal{L}^{new} = (x, T, S(x) - s_k)$ also by left-shifting: if $e < l$ then set $\mathcal{L}_{e+1} := \mathcal{L}^{new}$; otherwise set $\mathcal{L}_i := \mathcal{L}_{i+1}$ for $i = 1, \ldots, l-1$ and $\mathcal{L}_l := \mathcal{L}^{new}$. Cycling on the objective function value (line 20) is detected as follows: we check for the last $maxc.max\delta$ iterations where $maxc$ and $max\delta$ are two parameters to see if there is any period of length $\delta, 1 \leq \delta \leq \max \delta$, for which there are periodic repetitions of objective function values. For example, suppose $\max \delta = 3, \max c = 2$, and objective function values have been obtained in order $(1, 4, 5, 1, 2, 3, 1, 2, 3, 1, 2, 3)$ for the last 12 iterations, then there is a repetition period $\delta = 3$ (observe the last 9 values).

The second generic TS algorithm to be presented is *Tabu Search with Alternating Phases* (*TSAP*). This is based on the original idea proposed by Glover [Ree93], which has been largely unattempted so far. Two alternating phases in TSAP are the *improving* phase (Phase 1) and the *diversifying* phase (Phase 2). Both phases apply the *first-fit* principle, which makes the first improving move encountered. In the first phase, the algorithm inspects neighborhood $N_1(x)$ (or equivalently the list of applicable moves $S_1(x)$). The first improving move found (if any) is executed, then the TSAP restarts Phase 1 with the newly found solution as the new starting

solution. If no improving move is found in Phase 1, the algorithm switches to Phase 2 to diversify the search. In this phase, the algorithm searches another neighborhood $N_2(x)$ given by a move list $S_2(x)$ and with the help of a tabu list T. $N_2(x)$ is usually different from $N_1(x)$. If an improving move in $N_2(x)$ is found, it is made and the search restarts from Phase 1 with the found solution, else the best non-tabu move is selected and used as the new initial solution in the next search starting in Phase 1. The TSBJT stops when a certain criterion is met, e.g. the maximum number of iterations in both phases ($\max iter$) is reached; or the maximum number of consecutive non-improving moves in Phase 2 ($\max q$) is reached; or the allocated time is over; or objective function value's cycling is detected, etc.

The pseudo-code of TSAP is given below. Notations s_k^i and s_k^{cand} respectively stand for move i in iteration k and the best candidate move in iteration k.

algorithm *TSAP*

begin

Calculate an initial feasible solution x.

Set the best solution so far $x^* := x$ and the best objective function value $c^* = c(x^*)$.

Set tabu list $T := \emptyset$, the iteration counter $k := 0$, and the non-improving counter $q := 0$.

while $k < \max iter$ and $q < \max q$ and the allotted time is not over **do**

Phase 1:

Set $k := k + 1$.

Find a candidate list of moves $S_1(x)$.

if $S_1(x) = \emptyset$ **then goto** Phase 2.

for $i = 1$ to $|S_1(x)|$ **do**

if s_k^i is an improving move, **then**

Set $x := s_k^i(x)$, $x^* := x$, $c^* := c(x)$, $q := 0$, **goto** *Phase 1.*

end (for)

Phase 2:

Find a candidate list of moves $S_2(x)$.

if $S_2(x) = \emptyset$ **then stop**.

Initialize $s_k^{cand} := \emptyset$, $c^{cand} := \infty$.

for $i = 1$ to $|S_2(x)|$ **do**

if s_k^i is an improving move, **then**

Set $x := s_k^i(x)$, $x^* := x$, $c^* := c(x)$, $q := 0$, **goto** *Phase 1.*

if $s_k^i \notin T$ and $c(s_k^i(x)) < c^{cand}$ **then**

21 Set $s_k^{cand} := s_k^i$, $c^{cand} := c(s_k^i(x))$.

22 **end** (for)

23 **if** $s_k^{cand} \neq \emptyset$ **then** $x := s_k^{cand}(x)$.

24 **else** s_k^{cand} := the oldest tabu move in $T \cap S_2(x)$.

25 Set $x := s_k^{cand}(x)$, calculate $c(x)$.

26 **if** repetition on $c(x)$ is detected **then stop**.

27 Update T with the reverse move $(s_k^{cand})^{-1}$.

28 Set $q := q + 1$. Update the running time so far.

29 **end** (while)

end (*TSAP*)

The last generic TS to study is *Tabu Search with Variable Neighborhoods (TSVN)*. This algorithm inspects different neighborhoods systematically in the Tabu Search framework. This systematic use of neighborhood is inspired by the neighborhood exploration approach in *Variable Neighborhood Search* [HM01]. The TSVN algorithm is outlined as follows. Given a set of y different neighborhood structures $N_1, \ldots, N_y$ and their associated tabu lists $T_1, \ldots, T_y$, the TSVN sequentially searches neighborhoods $N_r(x), 1 \leq r \leq y$ according to the first-fit principle. The algorithm executes the first improving move (if any), updates the neighborhood's associated tabu list, and then starts over from neighborhood N_1 with the newly found solution used as the starting solution. If a neighborhood contains no improving move then the TSVN records the best non-tabu move of this neighborhood and switches to the next one. If all neighborhoods have been inspected without finding any improving move then all best non-tabu moves recorded are compared in terms of the objective function value and the best of them is selected. After making this selected move and updating the corresponding tabu list, the algorithm continues until some stopping criterion is met (e.g. the maximum number of iterations is reached; or the maximum number of consecutive non-improving moves is reached; or the allocated time is over; or objective function value's cycling is detected, etc).

Detailed steps of the TSVN are given below. In each iteration, neighborhood $N_r(x)$ is given through the corresponding candidate list of moves $S_r(x)$. Each move is denoted as s_{ik}^r where r is the neighborhood structure index, i is the order of the move in its neighborhood, and k is the iteration. For each neighborhood $r, 1 \leq r \leq y$,denote respectively by s^r, T_r, and c^r the first non-tabu move, the tabu list, and the move's corresponding objective function value. c^{cand} is the best move among all non-tabu non-improving moves found in each iteration.

algorithm *TSVN*

begin

1 Calculate an initial feasible solution x.

Set the best solution so far $x^* := x$ and the best objective function value $c^* = c(x^*)$.
Set the iteration counter $k := 0$ and the non-improving counter $q := 0$.
Set tabu list $T_i := \emptyset$, $1 \leq i \leq q$.
Loop 1:
while $k <$ max *iter* and $q <$ max q and the allotted running time is not over **do**
    Initialize $c^{cand} := \infty$.
    Set the neighborhood counter $r := 1$.
    Loop 2:
    while $r \leq y$ **do**
        Initialize $s^r := \emptyset$, $c^r := \infty$.
        Find a candidate list of moves $S_r(x)$.
        if $S_r(x) = \emptyset$ **then** set $r := r + 1$, **goto** *Loop 2*.
        for $i = 1$ to $|S_r(x)|$ **do**
            if s^r_{ik} is an improving move, **then**
                Set $x := s^r_{ik}(x)$, $x^* := x$, $c^* := c(x), q := 0$.
                Update T_r with the reverse move $(s^r_{ik})^{-1}$ and **goto** *Loop 1*.
            end (if)
            if $s^r_{ik} \notin T_r$ and $c(s^r_{ik}(x)) < c^r$ **then** set $s^r := s^r_{ik}$, $c^r := c(s^r_{ik}(x))$.
        end (for)
        if $c^{cand} < c^r$ **then** set $c^{cand} := c^r$.
        Set $r := r + 1$.
    end (while Loop 2)
    if $c^{cand} = c^p, 1 \leq p \leq y$, **then**
        Set $x := s^p(x)$, $x^* := x$, $c^* := c(x)$.
        Update T_p with the reverse move $(s^p)^{-1}$.
    end (if)
    if repetition on $c(x)$ is detected **then stop**.
    Set $q := q + 1$. Update the running time so far.
end (while Loop 1)
end (*TSAP*)

3.8.2 Tabu search algorithms for the FGBJS

Combining three generic TS algorithms *TSBJT*, *TSAP* and *TSVN* with three neighborhoods: simple $N^s(x)$, pairwise exchange $N^p(x)$, and flexible closure $N^c(x)$ results in the following six TS algorithms for the FGBJS:

1. TSBJT with neighborhoods $N^s(x)$ and $N^p(x)$ combined ($TSBJT - N^s/N^p$),
2. TSBJT with neighborhood $N^c(x)$($TSBJT - N^c$),
3. TSAP with neighborhood $N^p(x)$ in phase 1 and neighborhood $N^s(x)$ in phase 2 ($TSAP - N^p - N^s$),
4. TSAP with neighborhood $N^s(x)$ in phase 1 and neighborhood $N^c(x)$ in phase 2 ($TSAP - N^s - N^c$),
5. TSAP with neighborhood $N^p(x)$ in phase 1 and neighborhood $N^c(x)$ in phase 2 ($TSAP - N^p - N^c$),
6. TSVN with neighborhoods $N^s(x)$, $N^p(x)$ and $N^c(x)$ to be inspected in this sequence ($TSVN - N^s - N^p - N^c$).

The first two algorithms are based on the generic algorithm TSBJT. Algorithm $TSBJT - N^s/N^p$ is developed from a Glover's proposal to combine different move types [Ree93]. It uses the 4-tuple (i, u, v, k) format to present both simple and pairwise exchange moves. When evaluating or executing a move (i, j, v, k), the algorithm checks if j is the operation that immediately precedes ij in sequence $\pi_{\mu(j)}$. If yes, the algorithm makes the pairwise exchange move (j, i); otherwise it carries out an appropriate simple move (i, j, v, k). Algorithms (3)(4)(5) are based on the generic TS algorithm TSAP. Algorithms $TSAP - N^v - N^c$ and $TSAP - N^p - N^c$ use the flexible closure neighborhood in their diversifying phase. When considering all moves associated with all critical operations, the size of this neighborhood becomes very large and its full inspection is very time-consuming. For the purpose of diversifying the search, it is sufficient to search in a reduced flexible closure neighborhood that considers only the first operation of each job block belonging to the critical path. This neighborhood can be reduced even further when considering only moves (j, i) where j starts earlier than i in the current solution. Similarly, the full flexible closure neighborhood used in the $TSBJT - N^c$ is huge; hence the search can be limited to its reduced version (which is still very large). The last algorithm is based on the TSVN.

3.9 Computational results

3.9.1 Benchmark instances

Three constructive heuristics introduced in Section 3.7 and six TS algorithms presented in Section 3.8 were coded in C++ and tested on 160 benchmark instances on a PC having a processor Pentium IV 2.8 GHz and 512 MB of memory. Initially, no benchmark instances were available since the FGBJS has never been addressed in scheduling literature. However, it is possible to create FGBJS instances from available benchmark instances proposed for the FJS and the GBJS. First, we chose 160 FJS instances generated by Hurink, Jurisch and Thole. These instances are evenly divided in four instance groups, namely *sdata, edata, rdata,* and *vdata* [HJT94]; each group consists of Lawrence's 40 JS base instances (labeled *la01-la40)* modified with a certain level of flexibility. The groups' flexibility levels are in the order *sdata, edata, rdata,* and *vdata,* where instance set *sdata* is the original Lawrence's JS set without flexibility (see Table 3-5). In any instance, the default processor for any operation is its processor in the base JS instance. Because assignable processors for each operations were randomly generated, for any two instances derived from a same base JS instance but differing in their flexibility levels, e.g. *la01* in *edata* and *la01* in *vdata*, an assignable processor for an operation in one instance might not be a valid processor for this operation in the other instance which has a higher flexibility level.

Instance	Size		sdata		edata		rdata		vdata	
sets	n	m	*aveg*	*max*	*aveg*	*max*	*aveg*	*max*	*aveg*	*max*
la01-la05	10	5	1	1	1.15	2	2	3	2.5	4
la06-la10	15	5	1	1	1.15	2	2	3	2.5	4
la11-la15	20	5	1	1	1.15	2	2	3	2.5	4
la16-la20	10	10	1	1	1.15	3	2	3	5	8
la21-la25	15	10	1	1	1.15	3	2	3	5	8
la26-la30	20	10	1	1	1.15	3	2	3	5	8
la31-la35	30	10	1	1	1.15	3	2	3	5	8
la36-la40	15	15	1	1	1.15	3	2	3	7.5	12

n : number of jobs, m : number of processors
aveg: average number of assignable processors for each operation
max: maximum number of assignable processors for each operation

Table 3-5: Flexibility levels in FJS benchmark instances.

We then added to the above FJS instances transfer and sequence-dependent setup times taken from GBJS benchmark instances developed by Gröflin and Klinkert [GK05]. These GBJS instances were modified from the same Lawrence's base instances with transfer and sequence-dependent setup times uniformly distributed in the intervals $[1, 20]$ and $[0, 50]$ respectively. For simplicity, the setup times are processor-independent. The creation of the FGBJS instances was completed with 160 FGBJS instances divided in four groups. Each generated FGBJS instance can be referred to by its base JS instance's name and its flexibility group's name (e.g. *la01-sdata* stands for instance *la01* in *sdata* group).

The computational experiments conducted on these 160 FGBJS instances were carried out in three steps:

1. Evaluate three constructive heuristics SG-MOC, SG-MFP, and SG-BN.
2. Evaluate six TS algorithms with a common constructive heuristic to generate initial feasible schedules.
3. Evaluate the dependence of the TS algorithms on different initial solutions.

To experiment blocking's effects as much as possible, we did not allow any two consecutive operations of the same job to be performed by the same processor. Algorithms were evaluated by two criteria: efficiency and effectiveness. Efficiency measures computing effort (in CPU times) to obtain a solution. Effectiveness measures how good a solution is by comparing the solution's objective function value to a known optimal solution's objective function value or a known tight lower bound. The performance of the FGBJS's MILP formulation on these benchmark instances is, as anticipated, disappointing. Only 55 out of 160 instances, most of them in *sdata* set, could obtain feasible solutions after a 30-min of computing time per instance by CPLEX 9.0 solver (see Appendix). Obtained lower bounds were also of low quality. Further, since both the FGBJS problem and the solution methods are novel, no previous computational results exist to assert any particular solution. Therefore, we could only evaluate the algorithms by comparing their performances among themselves.

3.9.2 Computational results on the constructive heuristics

The performance of three heuristics SG-MOC, SG-MFP, and SG-BN on each data set *sdata, sdata, rdata,* and *vdata* are respectively presented in four tables from Table 3-6 to Table 3-9. Column (3) of these tables shows the makespans obtained by the permutation schedule corresponding to a non-decreasing order of total average times of jobs. Each entry of three columns (4)-(6) shows for each instance the makespan obtained by the corresponding heuristic (e.g. applying the $SG-MFP$ on the $la01-sdata$ yields makespan 1988, see line 1 of Table 3-6). For each instance, the best of three obtained makespans is shown in column (7). Columns (8)-(10) show the derived makespan deviations calculated for each makespan $C_{\max}$ as $\frac{C_{\max}- C^{best}}{C^{best}} \times 100\%$ where C^{best} is the corresponding best makespan obtained from the information in columns (4)-(7). The last four columns (11)-(13) show the heuristics' running times in seconds.

Instance	Size		Makespan of SG-				Deviation (%) of SG-			Time (seconds) of SG-		
lax-sdata	nxm	Seq	MOC	MFP	BN	Best	MOC	MFP	BN	MOC	MFP	BN
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)
la01-sdata	10x5	2925	1737	1988	2368	1737	0.00	14.45	36.33	3.23	2.35	2.07
la02-sdata	10x5	2779	1949	1841	2106	1841	5.87	0.00	14.39	3.25	2.25	3.21
la03-sdata	10x5	2614	1745	1542	1988	1542	13.16	0.00	28.92	3.29	2.38	2.23
la04-sdata	10x5	2430	1917	1994	1952	1917	0.00	4.02	1.83	3.52	2.44	2.17
la05-sdata	10x5	2809	1470	1696	1878	1470	0.00	15.37	27.76	3.07	2.66	2.11
la06-sdata	15x5	4181	2577	2472	3004	2472	4.25	0.00	21.52	7.21	3.66	4.53
la07-sdata	15x5	3870	2684	2526	2751	2526	6.25	0.00	8.91	7.29	3.91	4.49
la08-sdata	15x5	4089	2738	2406	2518	2406	13.80	0.00	4.66	7.12	3.79	4.74
la09-sdata	15x5	4012	2696	2717	2944	2696	0.00	0.78	9.20	7.19	3.81	4.45
la10-sdata	15x5	3764	2629	2742	3428	2629	0.00	4.30	30.39	6.82	3.57	4.50
la11-sdata	20x5	5264	3653	3526	3461	3461	5.55	1.88	0.00	12.70	5.65	8.41
la12-sdata	20x5	5202	3418	3218	3464	3218	6.22	0.00	7.64	12.17	5.17	8.03
la13-sdata	20x5	5228	3373	3566	3830	3373	0.00	5.72	13.55	13.12	5.26	8.27
la14-sdata	20x5	5456	3523	3446	3872	3446	2.23	0.00	12.36	13.11	5.62	7.95
la15-sdata	20x5	5446	3368	3399	3667	3368	0.00	0.92	8.88	12.13	5.41	7.72
la16-sdata	10x10	5387	3091	2886	3779	2886	7.10	0.00	30.94	5.77	7.19	3.30
la17-sdata	10x10	4991	2818	2430	3451	2430	15.97	0.00	42.02	5.72	6.58	3.59
la18-sdata	10x10	5385	2964	3098	3349	2964	0.00	4.52	12.99	5.73	6.85	3.68
la19-sdata	10x10	5896	2887	2685	3567	2685	7.52	0.00	32.85	5.67	6.82	3.39
la20-sdata	10x10	5293	3705	2969	3644	2969	24.79	0.00	22.73	5.88	6.84	3.48
la21-sdata	15x10	7309	4720	4219	4975	4219	11.87	0.00	17.92	13.14	11.82	7.28
la22-sdata	15x10	7251	4954	3880	4561	3880	27.68	0.00	17.55	13.63	11.90	7.41
la23-sdata	15x10	8518	4901	4528	5295	4528	8.24	0.00	16.94	13.82	11.43	7.33
la24-sdata	15x10	7990	4242	4231	4645	4231	0.26	0.00	9.78	13.84	11.28	7.68
la25-sdata	15x10	8163	4693	4059	5187	4059	15.62	0.00	27.79	12.69	11.17	7.34
la26-sdata	20x10	10161	5996	4752	6119	4752	26.18	0.00	28.77	26.16	14.56	14.39
la27-sdata	20x10	10548	5388	5600	6985	5388	0.00	3.93	29.64	25.13	16.15	12.35
la28-sdata	20x10	10412	6341	5723	6601	5723	10.80	0.00	15.34	21.94	9.99	8.07
la29-sdata	20x10	10940	6040	5313	6228	5313	13.68	0.00	17.22	17.79	10.00	8.27
la30-sdata	20x10	10690	6198	5247	6596	5247	18.12	0.00	25.71	15.38	9.94	7.67
la31-sdata	30x10	15535	8654	7916	9155	7916	9.32	0.00	15.65	70.36	25.16	25.28
la32-sdata	30x10	16343	9114	8067	9632	8067	12.98	0.00	19.40	63.99	31.89	24.97
la33-sdata	30x10	15449	8251	8417	9589	8251	0.00	2.01	16.22	65.01	28.07	22.98
la34-sdata	30x10	15614	7976	8720	8832	7976	0.00	9.33	10.73	66.43	24.39	23.52
la35-sdata	30x10	14392	8605	7974	10122	7974	7.91	0.00	26.94	64.94	27.44	23.37
la36-sdata	15x15	11925	6587	5708	7714	5708	15.40	0.00	35.14	12.00	14.70	6.56
la37-sdata	15x15	12423	7143	5905	8229	5905	20.97	0.00	39.36	13.21	15.28	7.71
la38-sdata	15x15	11242	6415	5379	8038	5379	19.26	0.00	49.43	10.96	13.59	6.83
la39-sdata	15x15	11891	5603	5901	7331	5603	0.00	5.32	30.84	11.27	14.19	7.07
la40-sdata	15x15	12493	6572	5310	7767	5310	23.77	0.00	46.27	12.63	12.57	6.72

Table 3-6: Performance of the constructive heuristics on *sdata* set.

Instance	Size		Makespan of SG-				Deviation (%) of SG-			Time (seconds) of SG-		
lax-edata	nxm	Seq	MOC	MFP	BN	Best	MOC	MFP	BN	MOC	MFP	BN
la01-edata	10x5	2925	2320	1909	2294	1909	21.53	0.00	20.17	1.06	0.70	0.85
la02-edata	10x5	2779	1916	1795	2060	1795	6.74	0.00	14.76	1.14	0.78	0.77
la03-edata	10x5	2614	1855	1666	1721	1666	11.34	0.00	3.30	1.09	0.78	0.84
la04-edata	10x5	2430	1799	1782	1760	1760	2.22	1.25	0.00	1.14	0.78	0.78
la05-edata	10x5	2809	1504	1740	1796	1504	0.00	15.69	19.41	1.10	0.91	0.78
la06-edata	15x5	4181	2605	2611	3189	2605	0.00	0.23	22.42	2.66	1.79	1.76
la07-edata	15x5	3870	2447	2429	2877	2429	0.74	0.00	18.44	2.61	1.46	1.89
la08-edata	15x5	4089	2702	2385	2518	2385	13.29	0.00	5.58	2.62	1.54	1.65
la09-edata	15x5	4012	2803	2709	2569	2569	9.11	5.45	0.00	2.71	1.55	1.93
la10-edata	15x5	3764	2835	2508	2822	2508	13.04	0.00	12.52	2.68	1.44	1.85
la11-edata	20x5	5264	3586	3500	3813	3500	2.46	0.00	8.94	5.22	2.51	3.74
la12-edata	20x5	5202	3333	3097	3347	3097	7.62	0.00	8.07	5.30	2.45	3.44
la13-edata	20x5	5228	3228	3633	3722	3228	0.00	12.55	15.30	5.13	2.74	3.52
la14-edata	20x5	5456	3543	3421	3611	3421	3.57	0.00	5.55	5.52	2.63	3.60
la15-edata	20x5	5446	3374	3274	3556	3274	3.05	0.00	8.61	5.44	2.33	3.88
la16-edata	10x10	5387	3392	2994	3847	2994	13.29	0.00	28.49	2.12	2.84	1.61
la17-edata	10x10	4991	2708	2462	3107	2462	9.99	0.00	26.20	2.05	2.77	1.53
la18-edata	10x10	5385	2886	3027	3317	2886	0.00	4.89	14.93	2.53	2.67	1.80
la19-edata	10x10	5896	3306	3035	3395	3035	8.93	0.00	11.86	2.22	2.79	1.76
la20-edata	10x10	5293	3551	2773	3780	2773	28.06	0.00	36.31	2.25	2.53	1.49
la21-edata	15x10	7309	4630	4235	5036	4235	9.33	0.00	18.91	6.74	5.04	4.08
la22-edata	15x10	7251	4223	4109	4842	4109	2.77	0.00	17.84	6.53	6.02	4.55
la23-edata	15x10	8518	4284	4167	5001	4167	2.81	0.00	20.01	6.94	5.95	4.66
la24-edata	15x10	7990	4429	4416	4951	4416	0.29	0.00	12.12	7.25	5.91	5.05
la25-edata	15x10	8163	4185	4182	4824	4182	0.07	0.00	15.35	6.45	5.87	4.03
la26-edata	20x10	10161	5544	5270	6365	5270	5.20	0.00	20.78	14.57	10.67	8.40
la27-edata	20x10	10548	5962	5422	6975	5422	9.96	0.00	28.64	13.91	10.96	8.78
la28-edata	20x10	10412	5302	5094	6466	5094	4.08	0.00	26.93	14.29	9.92	8.22
la29-edata	20x10	10940	6065	5083	6389	5083	19.32	0.00	25.69	16.36	9.78	8.60
la30-edata	20x10	10690	6215	5611	6589	5611	10.76	0.00	17.43	14.93	9.88	8.39
la31-edata	30x10	15535	7982	8258	9788	7982	0.00	3.46	22.63	55.06	28.55	27.12
la32-edata	30x10	16343	9190	9124	10003	9124	0.72	0.00	9.63	52.30	30.00	23.63
la33-edata	30x10	15449	7951	8444	8882	7951	0.00	6.20	11.71	49.07	27.84	24.61
la34-edata	30x10	15614	7896	8051	9094	7896	0.00	1.96	15.17	58.56	25.14	24.32
la35-edata	30x10	14392	8139	7905	9751	7905	2.96	0.00	23.35	51.92	23.78	23.56
la36-edata	15x15	11925	6373	5708	8030	5708	11.65	0.00	40.68	11.44	14.40	7.53
la37-edata	15x15	12423	6451	5869	8502	5869	9.92	0.00	44.86	12.08	14.33	8.61
la38-edata	15x15	11242	5544	5978	8220	5544	0.00	7.83	48.27	11.77	12.62	6.95
la39-edata	15x15	11891	6750	5818	6955	5818	16.02	0.00	19.54	12.64	14.21	8.15
la40-edata	15x15	12493	6564	5896	7547	5896	11.33	0.00	28.00	12.26	11.36	7.64

Table 3-7: Performance of the constructive heuristics on *edata* set.

Instance	Size		Makespan of SG-				Deviation (%) of SG-			Time (seconds) of SG-		
lax-rdata	nxm	Seq	MOC	MFP	BN	Best	MOC	MFP	BN	MOC	MFP	BN
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)
la01-rdata	10x5	2925	2206	1908	2071	1908	15.62	0.00	8.54	1.06	0.79	0.96
la02-rdata	10x5	2779	2042	1944	1812	1812	12.69	7.28	0.00	1.11	0.85	1.16
la03-rdata	10x5	2614	1670	1527	1572	1527	9.36	0.00	2.95	1.17	1.00	0.91
la04-rdata	10x5	2430	1637	1653	1472	1472	11.21	12.30	0.00	1.26	0.85	0.97
la05-rdata	10x5	2809	1611	1664	1645	1611	0.00	3.29	2.11	1.19	0.94	0.89
la06-rdata	15x5	4181	2647	2765	2973	2647	0.00	4.46	12.32	2.76	1.91	2.05
la07-rdata	15x5	3870	2451	2706	2568	2451	0.00	10.40	4.77	2.90	1.66	2.25
la08-rdata	15x5	4089	2459	2208	2656	2208	11.37	0.00	20.29	3.01	1.61	2.31
la09-rdata	15x5	4012	2464	2681	2597	2464	0.00	8.81	5.40	2.75	1.65	2.04
la10-rdata	15x5	3764	2554	2637	2810	2554	0.00	3.25	10.02	2.74	1.63	2.26
la11-rdata	20x5	5264	3296	3224	3774	3224	2.23	0.00	17.06	5.49	2.93	4.23
la12-rdata	20x5	5202	3321	3181	3038	3038	9.32	4.71	0.00	5.74	3.13	4.26
la13-rdata	20x5	5228	3166	3297	3191	3166	0.00	4.14	0.79	5.83	2.67	4.64
la14-rdata	20x5	5456	3492	3227	3454	3227	8.21	0.00	7.03	5.64	3.14	4.05
la15-rdata	20x5	5446	3469	3566	3735	3469	0.00	2.80	7.67	5.88	2.95	4.47
la16-rdata	10x10	5387	3154	2891	3568	2891	9.10	0.00	23.42	2.47	3.01	2.18
la17-rdata	10x10	4991	2857	2456	3362	2456	16.33	0.00	36.89	2.53	2.88	2.22
la18-rdata	10x10	5385	3533	2666	3231	2666	32.52	0.00	21.19	2.80	2.87	2.20
la19-rdata	10x10	5896	2892	2851	3290	2851	1.44	0.00	15.40	2.63	3.08	2.17
la20-rdata	10x10	5293	3384	2911	3669	2911	16.25	0.00	26.04	2.59	2.93	2.08
la21-rdata	15x10	7309	4557	4322	4610	4322	5.44	0.00	6.66	7.86	6.87	5.70
la22-rdata	15x10	7251	3933	3326	4370	3326	18.25	0.00	31.39	7.41	5.89	5.48
la23-rdata	15x10	8518	4499	4057	4982	4057	10.89	0.00	22.80	8.18	6.59	5.81
la24-rdata	15x10	7990	4218	3737	4704	3737	12.87	0.00	25.88	7.03	6.37	5.74
la25-rdata	15x10	8163	3840	3476	4540	3476	10.47	0.00	30.61	7.04	6.19	6.00
la26-rdata	20x10	10161	5306	5273	5914	5273	0.63	0.00	12.16	16.88	13.10	12.56
la27-rdata	20x10	10548	5798	5476	6204	5476	5.88	0.00	13.29	17.19	14.42	14.31
la28-rdata	20x10	10412	5989	4947	6222	4947	21.06	0.00	25.77	16.42	13.60	12.48
la29-rdata	20x10	10940	5538	5306	6303	5306	4.37	0.00	18.79	16.91	14.10	12.74
la30-rdata	20x10	10690	5746	5519	6165	5519	4.11	0.00	11.71	17.76	12.95	12.11
la31-rdata	30x10	15535	8213	7868	9345	7868	4.38	0.00	18.77	52.97	41.18	36.61
la32-rdata	30x10	16343	8665	8436	10002	8436	2.71	0.00	18.56	59.33	37.11	37.29
la33-rdata	30x10	15449	7703	8031	8335	7703	0.00	4.26	8.20	50.35	37.18	34.32
la34-rdata	30x10	15614	8375	7698	8012	7698	8.79	0.00	4.08	54.98	38.91	37.79
la35-rdata	30x10	14392	8319	7952	8464	7952	4.62	0.00	6.44	52.26	37.11	36.08
la36-rdata	15x15	11925	5844	5833	7262	5833	0.19	0.00	24.50	13.70	14.72	13.79
la37-rdata	15x15	12423	6609	5662	7988	5662	16.73	0.00	41.08	14.06	15.39	11.41
la38-rdata	15x15	11242	5896	5613	7310	5613	5.04	0.00	30.23	13.30	17.90	12.23
la39-rdata	15x15	11891	6470	5595	7654	5595	15.64	0.00	36.80	13.39	18.45	12.15
la40-rdata	15x15	12493	6124	5691	7587	5691	7.61	0.00	33.32	14.09	15.20	12.34

Table 3-8: Performance of the constructive heuristics on *rdata* set.

Instance	Size		Makespan of SG-				Deviation (%) of SG-			Time (seconds) of SG-		
lax-vdata	nxm	Seq	MOC	MFP	BN	Best	MOC	MFP	BN	MOC	MFP	BN
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)
la01-vdata	10x5	2925	2004	1751	1894	1751	14.45	0.00	8.17	1.24	0.92	1.01
la02-vdata	10x5	2779	1634	1606	1886	1606	1.74	0.00	17.43	1.26	0.96	0.95
la03-vdata	10x5	2614	1743	1792	1643	1643	6.09	9.07	0.00	1.20	0.92	0.91
la04-vdata	10x5	2430	1548	1623	1658	1548	0.00	4.84	7.11	1.24	0.85	0.91
la05-vdata	10x5	2809	1423	1462	1961	1423	0.00	2.74	37.81	1.22	0.91	0.89
la06-vdata	15x5	4181	2594	2406	2629	2406	7.81	0.00	9.27	2.86	1.73	2.21
la07-vdata	15x5	3870	2494	2474	2442	2442	2.13	1.31	0.00	2.96	1.72	2.26
la08-vdata	15x5	4089	2316	2497	2342	2316	0.00	7.82	1.12	3.10	2.01	2.26
la09-vdata	15x5	4012	2864	2812	2734	2734	4.75	2.85	0.00	3.03	1.91	2.17
la10-vdata	15x5	3764	2644	2454	2471	2454	7.74	0.00	0.69	3.27	1.80	2.29
la11-vdata	20x5	5264	3319	3161	3303	3161	5.00	0.00	4.49	6.06	3.01	4.66
la12-vdata	20x5	5202	2944	3055	3054	2944	0.00	3.77	3.74	5.98	3.22	4.22
la13-vdata	20x5	5228	3159	2981	3318	2981	5.97	0.00	11.30	6.42	3.26	4.71
la14-vdata	20x5	5456	2882	2953	3411	2882	0.00	2.46	18.36	6.11	3.21	4.28
la15-vdata	20x5	5446	3244	3614	3440	3244	0.00	11.41	6.04	5.94	3.24	4.64
la16-vdata	10x10	5387	2568	2446	3019	2446	4.99	0.00	23.43	3.64	3.74	3.02
la17-vdata	10x10	4991	2524	2294	2712	2294	10.03	0.00	18.22	3.83	3.87	3.29
la18-vdata	10x10	5385	3052	2652	3028	2652	15.08	0.00	14.18	3.61	3.63	3.07
la19-vdata	10x10	5896	2623	2510	3280	2510	4.50	0.00	30.68	3.44	4.07	3.05
la20-vdata	10x10	5293	3119	2604	3309	2604	19.78	0.00	27.07	3.57	3.92	2.90
la21-vdata	15x10	7309	4474	3415	4463	3415	31.01	0.00	30.69	10.19	9.01	8.51
la22-vdata	15x10	7251	3859	3578	4302	3578	7.85	0.00	20.23	10.15	9.79	8.32
la23-vdata	15x10	8518	3803	4026	4688	3803	0.00	5.86	23.27	9.39	9.47	8.45
la24-vdata	15x10	7990	3512	3547	4391	3512	0.00	1.00	25.03	10.63	9.98	8.46
la25-vdata	15x10	8163	4106	4051	4600	4051	1.36	0.00	13.55	10.15	9.75	8.41
la26-vdata	20x10	10161	4991	4570	5449	4570	9.21	0.00	19.23	23.46	18.74	19.12
la27-vdata	20x10	10548	5266	5350	5972	5266	0.00	1.60	13.41	22.89	18.68	18.12
la28-vdata	20x10	10412	5080	4896	5514	4896	3.76	0.00	12.62	21.83	17.52	17.02
la29-vdata	20x10	10940	5062	4574	5792	4574	10.67	0.00	26.63	21.21	18.44	16.99
la30-vdata	20x10	10690	5646	4809	5283	4809	17.40	0.00	9.86	22.91	17.97	17.58
la31-vdata	30x10	15535	7753	6348	8393	6348	22.13	0.00	32.21	73.13	49.60	54.01
la32-vdata	30x10	16343	8356	7599	8005	7599	9.96	0.00	5.34	72.95	51.14	53.57
la33-vdata	30x10	15449	7371	7275	8854	7275	1.32	0.00	21.70	73.32	51.76	53.77
la34-vdata	30x10	15614	7246	6999	8186	6999	3.53	0.00	16.96	75.78	51.72	54.73
la35-vdata	30x10	14392	7487	7730	7595	7487	0.00	3.25	1.44	75.98	56.13	55.67
la36-vdata	15x15	11925	6017	4466	6540	4466	34.73	0.00	46.44	25.58	25.05	22.64
la37-vdata	15x15	12423	5661	5772	7592	5661	0.00	1.96	34.11	24.96	24.64	22.03
la38-vdata	15x15	11242	5257	4724	6171	4724	11.28	0.00	30.63	25.45	24.73	22.49
la39-vdata	15x15	11891	5724	4448	6846	4448	28.69	0.00	53.91	27.18	32.17	22.41
la40-vdata	15x15	12493	5739	5657	6676	5657	1.45	0.00	18.01	25.02	25.81	22.02

Table 3-9: Performance of the constructive heuristics on *vdata* set.

Data set	Average improvement (%)			Rank		
	SG-MOC	SG-MFP	SG-BN	SG-MOC	SG-MFP	SG-BN
sdata	39.83	43.39	33.00	2	1	3
edata	40.43	43.36	34.21	2	1	3
rdata	41.84	44.93	37.80	2	1	3
vdata	45.80	48.50	41.36	2	1	3
average	42.69	45.60	37.79	2	1	3

Table 3-10: Performance of the constructive heuristics w.r.t. improvement to permutation schedules.

Data set	Average deviation (%)			Rank		
	SG-MOC	SG-MFP	SG-BN	SG-MOC	SG-MFP	SG-BN
sdata	8.87	1.81	21.61	2	1	3
edata	6.80	1.49	18.71	2	1	3
rdata	7.88	1.64	16.07	2	1	3
vdata	7.61	1.50	17.36	2	1	3
average	7.79	1.61	18.44	2	1	3

Table 3-11: Performance of the constructive heuristics w.r.t. deviations.

Data set	# best makespans			Rank		
	SG-MOC	SG-MFP	SG-BN	SG-MOC	SG-MFP	SG-BN
sdata	12	27	1	2	1	3
edata	8	30	2	2	1	3
rdata	8	29	3	2	1	3
vdata	11	26	3	2	1	3
Total	39	112	9	2	1	3

Table 3-12: Performance of the constructive heuristics w.r.t. numbers of best makespans obtained.

Table 3-10 ranks the heuristics according to their average improvements to permutation schedules by non-decreasing order of total average times of jobs while Table 3-11 ranks them according to their deviations. Table 3-12 gives a ranking based on the number of times a heuristic obtained the best makespan. The outputs of these tables indicate that the SG-MFP heuristic is the best performing algorithm in terms of solution quality. As its average running time was only 11.52% higher than the average running time of the fastest heuristic (see Table 3-13), the effectiveness of SG-MFP heuristic did not require a significant trade-off of efficiency.

We further analyzed the impact of flexibility and instance size on the constructive heuristics' performance. First, the impact of flexibility was evaluated. We used the makespans associated with zero-flexibility instance set *sdata* as references to calculate the deviations of makespans associated with the flexible sets *edata*, *rdata*, and *vdata* (see Table 3-14). For instance with the SG-MFP, the makespan obtained on *la01-edata* (with flexibility) is 3.97% better than the one obtained on *la01-sdata* (without flexibility). Tables 3-14 and 3-15 suggest that in general, a higher degree of flexibility leads to a solution of better quality.

Table 3-16 presents the heuristics' average deviations and average computational times for each data set. Observe that: (1) when the ratio of the number of jobs to the number of processors ($n : m$) approached one, the average deviation increased and (2) when the value of the product $n \times m$ increased, the computational time also increased.

Data set	Average computing time (seconds) SG-MOC	SG-MFP	SG-BN
sdata	17.71	10.44	8.38
edata	12.29	8.01	6.66
rdata	13.07	10.39	9.63
vdata	18.30	14.52	14.20
average	15.34	10.84	9.72

Table 3-13: Computing times of the constructive heuristics.

3.9.3 Computational results of the Tabu search heuristics

We used the SG-MFP heuristic as an initial solution generator for six TS algorithms and carried out computational experiments for them on the same 160 FGBJS instances. The algorithms were run with a same set of Tabu parameters, including a maximum iteration number $\max iter = 10,000$, an universal tabu list length $t = 10$, a number of objective function value repetitions $\max c = 4$, and a checking length $\max \delta = 100$. Other stopping criteria include a maximum running time of 30 minutes and a maximum number of non-improving moves $\max q = 1000$. We obtained these parameter values from preliminary tuning. The TS algorithms were evaluated on their (relative) effectiveness as their running times were limited to the same value. We used the following three evaluation metrics: (1) the average deviation from the best makespans, (2) the number of times to obtain the best makespans, and (3) the average improvement made to initial solutions. An overall ranking of the TS algorithms was determined base on the average of these three metrics.

The algorithms' obtained makespans and performance measurements are presented in four tables from Table 3-20 to Table 3-22, corresponding to *sdata, edata, rdata,* and *vdata* sets, respectively. In each table, initial makespans are given in column (2) while makespans of the six algorithms are shown in columns (3) to (8). Columns (9)-(14) show for each instance how much its associated initial makespan was improved by a TS algorithm. Improvement is measured as $\frac{Initial\ makespan - Tabu\ search\ makespan}{Initial\ makespan} \times 100\%$. Column (15) shows for each instance the best objective function value among six values obtained by the algorithms. Deviations from the best makespans are given in the last six columns (16)-(21).

Tables 3-17 and 3-18 rank the performance of all six TS algorithms with respect to flexibility levels and instance sizes respectively.

The following remarks are made from Tables 3-17 and 3-18:

1. All algorithms made significant improvements to the initial makespans with average improvements ranging from 11.38% to 37.05%. On average, the initial makespans were improved by 27.94%.

2. No algorithm was the clear winner. On average, $TSBJT - N^s/N^p(A1)$ was the best and $TSBJT - N^c(A2)$ was the second-runner. However, the ranks varied over different

flexibility levels and different problem sizes. For instance, $TSBJT - N^s/N^p$ ranked first on *rdata* and *vdata* sets but only third on *sdata* and *sdata* sets; $TSAP - N^p - N^s(A3)$ trailed other algorithms most of time, but performed exceptionally well for *vdata* set which has the highest flexibility level.

3. The ranks based on different metrics were also not consistent. For example, $TSVN$ did well on *vdata* with respect to the average deviation metric but not to the other metrics.

4. Increasing flexibility gave better average improvement but did not always improve the other metrics.

Base	Size		MCO			MFP			BN	
inst.	nxm	edata	rdata	vdata	edata	rdata	vdata	edata	rdata	vdata
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)
la01	10x5	-33.56	-27.00	-15.37	3.97	4.02	11.92	3.13	3.13	20.02
la02	10x5	1.69	-4.77	16.16	2.50	-5.59	12.76	2.18	2.18	10.45
la03	10x5	-6.30	4.30	0.11	-8.04	0.97	-16.21	13.43	13.43	17.35
la04	10x5	6.16	14.61	19.25	10.63	17.10	18.61	9.84	9.84	15.06
la05	10x5	-2.31	-9.59	3.20	-2.59	1.89	13.80	4.37	4.37	-4.42
la06	15x5	-1.09	-2.72	-0.66	-5.62	-11.85	2.67	-6.16	-6.16	12.48
la07	15x5	8.83	8.68	7.08	3.84	-7.13	2.06	-4.58	-4.58	11.23
la08	15x5	1.31	10.19	15.41	0.87	8.23	-3.78	0.00	0.00	6.99
la09	15x5	-3.97	8.61	-6.23	0.29	1.32	-3.50	12.74	12.74	7.13
la10	15x5	-7.84	2.85	-0.57	8.53	3.83	10.50	17.68	17.68	27.92
la11	20x5	1.83	9.77	9.14	0.74	8.56	10.35	-10.17	-10.17	4.57
la12	20x5	2.49	2.84	13.87	3.76	1.15	5.07	3.38	3.38	11.84
la13	20x5	4.30	6.14	6.34	-1.88	7.54	16.40	2.82	2.82	13.37
la14	20x5	-0.57	0.88	18.19	0.73	6.36	14.31	6.74	6.74	11.91
la15	20x5	-0.18	-3.00	3.68	3.68	-4.91	-6.33	3.03	3.03	6.19
la16	10x10	-9.74	-2.04	16.92	-3.74	-0.17	15.25	-1.80	-1.80	20.11
la17	10x10	3.90	-1.38	10.43	-1.32	-1.07	5.60	9.97	9.97	21.41
la18	10x10	2.63	-19.20	-2.97	2.29	13.94	14.40	0.96	0.96	9.58
la19	10x10	-14.51	-0.17	9.14	-13.04	-6.18	6.52	4.82	4.82	8.05
la20	10x10	4.16	8.66	15.82	6.60	1.95	12.29	-3.73	-3.73	9.19
la21	15x10	1.91	3.45	5.21	-0.38	-2.44	19.06	-1.23	-1.23	10.29
la22	15x10	14.76	20.61	22.10	-5.90	14.28	7.78	-6.16	-6.16	5.68
la23	15x10	12.59	8.20	22.40	7.97	10.40	11.09	5.55	5.55	11.46
la24	15x10	-4.41	0.57	17.21	-4.37	11.68	16.17	-6.59	-6.59	5.47
la25	15x10	10.82	18.18	12.51	-3.03	14.36	0.20	7.00	7.00	11.32
la26	20x10	7.54	11.51	16.76	-10.90	-10.96	3.83	-4.02	-4.02	10.95
la27	20x10	-10.65	-7.61	2.26	3.18	2.21	4.46	0.14	0.14	14.50
la28	20x10	16.39	5.55	19.89	10.99	13.56	14.45	2.05	2.05	16.47
la29	20x10	-0.41	8.31	16.19	4.33	0.13	13.91	-2.59	-2.59	7.00
la30	20x10	-0.27	7.29	8.91	-6.94	-5.18	8.35	0.11	0.11	19.91
la31	30x10	7.77	5.10	10.41	-4.32	0.61	19.81	-6.91	-6.91	8.32
la32	30x10	-0.83	4.93	8.32	-13.10	-4.57	5.80	-3.85	-3.85	16.89
la33	30x10	3.64	6.64	10.67	-0.32	4.59	13.57	7.37	7.37	7.67
la34	30x10	1.00	-5.00	9.15	7.67	11.72	19.74	-2.97	-2.97	7.31
la35	30x10	5.42	3.32	12.99	0.87	0.28	3.06	3.67	3.67	24.97
la36	15x15	3.25	11.28	8.65	0.00	-2.19	21.76	-4.10	-4.10	15.22
la37	15x15	9.69	7.48	20.75	0.61	4.12	2.25	-3.32	-3.32	7.74
la38	15x15	13.58	8.09	18.05	-11.14	-4.35	12.18	-2.26	-2.26	23.23
la39	15x15	-20.47	-15.47	-2.16	1.41	5.19	24.62	5.13	5.13	6.62
la40	15x15	0.12	6.82	12.67	-11.04	-7.18	-6.53	2.83	2.83	14.05

Table 3-14: Impact of flexibility on performance of the constructive heuristics.

Base	MCO				MFP				MFP			
instance	s	e	r	v	s	e	r	v	s	e	r	v
la01-la40	3	2	6	29	1	2	6	31	0	1	9	30
s : sdata, e : edata, r : rdata, v : vdata												

Table 3-15: Impact of flexibility on the number of best makespans obtained.

Data set	Size n	Size m	Average deviation (%) MOC	MFP	BN	Average computing time (s) MOC	MFP	BN
la01-la05	10	5	11.27	9.49	11.87	1.69	1.25	1.27
la06-la10	15	5	5.71	5.11	10.07	3.92	2.21	2.70
la11-la15	20	5	7.64	7.61	6.95	7.44	3.53	5.13
la16-la20	10	10	11.29	9.21	10.56	3.55	4.09	2.59
la21-la25	15	10	10.78	12.07	6.06	9.45	8.31	6.51
la26-la30	20	10	9.02	7.44	10.90	18.90	13.57	12.31
la31-la35	30	10	6.85	10.80	11.71	61.93	36.20	34.86
la36-la40	15	15	9.75	13.47	11.14	15.85	17.56	12.36

Table 3-16: Impact of instance size on the performance of the constructive heuristics.

	TS algorithms A1	A2	A3	A4	A5	A6	Rank of algorithms A1	A2	A3	A4	A5	A6
Overall												
Avg. Imp.(%)	26.40	25.18	22.25	23.66	22.12	24.02	*1*	*2*	*5*	*4*	*6*	*3*
♯ best Cmax	51	32	14	31	22	16	*1*	*2*	*6*	*3*	*4*	*5*
Avg dev. (%)	5.13	7.23	9.94	10.31	11.41	7.93	*1*	*2*	*4*	*5*	*6*	*3*
Avg. ranks							*1.00*	*2.00*	*5.00*	*4.00*	*5.33*	*3.67*
sdata												
Avg. Imp.(%)	16.85	17.93	11.38	17.32	16.52	13.96	*3*	*1*	*6*	*2*	*4*	*5*
♯ best Cmax	9	15	1	6	12	3	*3*	*1*	*6*	*4*	*2*	*5*
Avg dev. (%)	4.21	2.92	11.30	3.58	4.63	8.00	*3*	*1*	*6*	*2*	*4*	*5*
Avg. ranks							*3.00*	*1.00*	*6.00*	*2.67*	*3.33*	*5.00*
edata												
Avg. Imp.(%)	19.52	20.67	14.51	20.44	18.34	17.86	*3*	*1*	*6*	*2*	*4*	*5*
♯ best Cmax	7	12	0	11	8	2	*4*	*1*	*6*	*2*	*3*	*5*
Avg dev. (%)	5.80	4.10	12.37	4.48	7.15	8.05	*3*	*1*	*6*	*2*	*4*	*5*
Avg. ranks							*3.33*	*1.00*	*6.00*	*2.00*	*3.67*	*5.00*
rdata												
Avg. Imp.(%)	32.17	28.45	26.81	28.75	24.04	28.05	*1*	*3*	*5*	*2*	*6*	*4*
♯ best Cmax	21	2	5	6	2	4	*1*	*5*	*3*	*2*	*5*	*4*
Avg dev. (%)	1.89	7.49	10.03	7.19	14.40	8.32	*1*	*3*	*5*	*2*	*6*	*4*
Avg. ranks							*1.00*	*3.67*	*4.33*	*2.00*	*5.67*	*4.00*
vdata												
Avg. Imp.(%)	37.05	33.68	36.30	28.11	29.60	36.19	*1*	*4*	*2*	*6*	*5*	*3*
♯ best Cmax	14	3	8	8	0	7	*1*	*5*	*2*	*2*	*6*	*4*
Avg dev. (%)	8.82	14.24	8.37	26.39	21.30	8.70	*3*	*4*	*2*	*6*	*5*	*1*
Avg. ranks							*1.67*	*4.33*	*2.00*	*4.67*	*5.33*	*2.67*

Avg. Imp.(%) - average improvement (in %)
♯ best Cmax - number of times an algorithm obtained the best makespan
Avg dev. (%) - average deviation from the best makespan in %
Avg. ranks - average ranks of the algorithms
$A1 : TSBJT - N^s/N^p$, $A2 : TSBJT - N^c$, $A3 : TSAP - N^p - N^s$,
$A4 : TSBJT - N^s - N^c$, $A5 : TSBJT - N^p - N^c$, $A6 : TSAP - N^s - N^p - N^c$

Table 3-17: Ranks of the TS algorithms with respect to flexibility levels.

	TS algorithms						Rank of algorithms					
	A1	A2	A3	A4	A5	A6	A1	A2	A3	A4	A5	A6
la01-la05 (10x5)												
Avg. Imp.(%)	21.12	21.74	14.06	19.19	13.87	16.52	*2*	*1*	*5*	*3*	*6*	*4*
♯ best Cmax	10	10	1	2	1	1	*1*	*1*	*4*	*3*	*4*	*4*
Avg dev. (%)	2.83	2.26	12.32	5.65	12.88	9.16	*2*	*1*	*5*	*3*	*6*	*4*
Avg. ranks							*1.67*	*1.00*	*4.67*	*3.00*	*5.33*	*4.00*
la06-la10 (15x5)												
Avg. Imp.(%)	19.42	18.21	13.07	17.90	14.86	14.83	*1*	*2*	*6*	*3*	*4*	*5*
♯ best Cmax	10	5	0	5	0	0	*1*	*2*	*4*	*2*	*4*	*4*
Avg dev. (%)	2.09	4.11	10.50	4.32	8.47	8.42	*1*	*2*	*6*	*3*	*5*	*4*
Avg. ranks							*1.00*	*2.00*	*5.33*	*2.67*	*4.33*	*4.33*
la11-la15 (20x5)												
Avg. Imp.(%)	16.60	13.47	11.45	16.75	13.59	14.04	*2*	*5*	*6*	*1*	*4*	*3*
♯ best Cmax	8	2	1	6	3	1	*1*	*4*	*5*	*2*	*3*	*5*
Avg dev. (%)	2.58	6.53	8.93	2.36	6.43	5.83	*2*	*5*	*6*	*1*	*4*	*3*
Avg. ranks							*1.67*	*4.67*	*5.67*	*1.33*	*3.67*	*3.67*
la16-la20 (10x10)												
Avg. Imp.(%)	34.19	31.77	25.23	30.10	23.28	26.60	*1*	*2*	*5*	*3*	*6*	*4*
♯ best Cmax	10	5	0	5	0	0	*1*	*2*	*4*	*2*	*4*	*4*
Avg dev. (%)	1.78	6.13	15.69	8.37	19.66	13.84	*1*	*2*	*5*	*3*	*6*	*4*
Avg. ranks							*1.00*	*2.00*	*4.67*	*2.67*	*5.33*	*4.00*
la21-la25 (15x10)												
Avg. Imp.(%)	31.75	29.35	25.81	32.16	26.58	28.58	*2*	*3*	*6*	*1*	*5*	*4*
♯ best Cmax	4	4	1	7	2	2	*2*	*2*	*6*	*1*	*4*	*4*
Avg dev. (%)	4.39	8.10	13.01	3.49	12.88	9.05	*2*	*3*	*6*	*1*	*5*	*4*
Avg. ranks							*2.00*	*2.67*	*6.00*	*1.00*	*4.67*	*4.00*
la26-la30 (20x10)												
Avg. Imp.(%)	28.25	26.38	25.57	23.16	25.88	26.78	*1*	*3*	*5*	*6*	*4*	*2*
♯ best Cmax	5	1	2	2	6	4	*2*	*6*	*4*	*4*	*1*	*3*
Avg dev. (%)	4.00	7.31	7.53	13.30	8.18	5.69	*1*	*3*	*4*	*6*	*5*	*2*
Avg. ranks							*1.33*	*4.00*	*4.33*	*5.33*	*3.33*	*2.33*
la31-la35 (30x10)												
Avg. Imp.(%)	24.40	24.11	27.05	19.98	24.38	29.22	*3*	*5*	*2*	*6*	*4*	*1*
♯ best Cmax	1	3	6	1	5	4	*5*	*4*	*1*	*5*	*2*	*3*
Avg dev. (%)	12.73	14.03	7.19	21.51	13.64	5.00	*3*	*5*	*2*	*6*	*4*	*1*
Avg. ranks							*3.67*	*4.67*	*1.67*	*5.67*	*3.33*	*1.67*
la36-la40 (15x15)												
Avg. Imp.(%)	35.46	36.43	35.77	30.01	34.55	35.55	*4*	*1*	*2*	*6*	*5*	*3*
♯ best Cmax	3	2	3	3	5	4	*3*	*6*	*3*	*3*	*1*	*2*
Avg dev. (%)	11.05	9.00	8.94	24.28	12.83	9.14	*4*	*2*	*1*	*6*	*5*	*3*
Avg. ranks							*3.67*	*3.00*	*2.00*	*5.00*	*3.67*	*2.67*

Avg. Imp.(%) - average improvement (in %)
♯ best Cmax - number of times an algorithm obtained the best makespan
Avg dev. (%) - average deviation from the best makespan (in %)
Avg. ranks - average ranks of the algorithms
$A1 : TSBJT - N^s/N^p$, $A2 : TSBJT - N^c$, $A3 : TSAP - N^p - N^s$,
$A4 : TSBJT - N^s - N^c$, $A5 : TSBJT - N^p - N^c$, $A6 : TSAP - N^s - N^p - N^c$

Table 3-18: Ranks of the TS algorithms with respect to instance sizes.

sdata	Makespan							Improvement (%)						Best	Deviation from the best (%)					
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)	(17)	(18)	(19)	(20)	(21)
Instance	Initial	A1	A2	A3	A4	A5	A6	A1	A2	A3	A4	A5	A6		A1	A2	A3	A4	A5	A6
la01-sdata	1988	1773	1583	1773	1631	1816	1773	10.81	20.37	10.81	17.96	8.65	10.81	1583	12.00	0.00	12.00	3.03	14.72	12.00
la02-sdata	1841	1570	1546	1751	1619	1619	1673	14.72	16.02	4.89	12.06	12.06	9.13	1546	1.55	0.00	13.26	4.72	4.72	8.21
la03-sdata	1542	1423	1423	1423	1423	1423	1423	7.72	7.72	7.72	7.72	7.72	7.72	1423	0.00	0.00	0.00	0.00	0.00	0.00
la04-sdata	1994	1510	1439	1575	1510	1546	1691	24.27	27.83	21.01	24.27	22.47	15.20	1439	4.93	0.00	9.45	4.93	7.44	17.51
la05-sdata	1696	1521	1430	1596	1482	1474	1476	10.32	15.68	5.90	12.62	13.09	12.97	1430	6.36	0.00	11.61	3.64	3.08	3.22
la06-sdata	2472	2333	2229	2369	2342	2360	2342	5.62	9.83	4.17	5.26	4.53	5.26	2229	4.67	0.00	6.28	5.07	5.88	5.07
la07-sdata	2526	2229	2146	2339	2166	2191	2194	11.76	15.04	7.40	14.25	13.26	13.14	2146	3.87	0.00	8.99	0.93	2.10	2.24
la08-sdata	2406	2234	2239	2390	2382	2396	2390	7.15	6.94	0.67	1.00	0.42	0.67	2234	0.00	0.22	6.98	6.62	7.25	6.98
la09-sdata	2717	2386	2301	2551	2277	2409	2409	12.18	15.31	6.11	16.19	11.34	11.34	2277	4.79	1.05	12.03	0.00	5.80	5.80
la10-sdata	2742	2591	2413	2612	2481	2582	2481	5.51	12.00	4.74	9.52	5.84	9.52	2413	7.38	0.00	8.25	2.82	7.00	2.82
la11-sdata	3526	3142	3065	3318	2961	2961	3144	10.89	13.07	5.90	16.02	16.02	10.83	2961	6.11	3.51	12.06	0.00	0.00	6.18
la12-sdata	3218	2982	3175	3189	3143	3143	3175	7.33	1.34	0.90	2.33	2.33	1.34	2982	0.00	6.47	6.94	5.40	5.40	6.47
la13-sdata	3566	3186	2862	3276	2877	2923	3176	10.66	19.74	8.13	19.32	18.03	10.94	2862	11.32	0.00	14.47	0.52	2.13	10.97
la14-sdata	3446	3148	3103	3343	3179	3086	3301	8.65	9.95	2.99	7.75	10.45	4.21	3086	2.01	0.55	8.33	3.01	0.00	6.97
la15-sdata	3399	3045	3146	3300	3000	2985	3079	10.41	7.44	2.91	11.74	12.18	9.41	2985	2.01	5.39	10.55	0.50	0.00	3.15
la16-sdata	2886	2303	2146	2351	2157	2367	2448	20.20	25.64	18.54	25.26	17.98	15.18	2146	7.32	0.00	9.55	0.51	10.30	14.07
la17-sdata	2430	2053	2083	2257	2116	2209	2187	15.51	14.28	7.12	12.92	9.09	10.00	2053	0.00	1.46	9.94	3.07	7.60	6.53
la18-sdata	3098	2234	2227	2552	2656	2326	2504	27.89	28.11	17.62	14.27	24.92	19.17	2227	0.31	0.00	14.59	19.26	4.45	12.44
la19-sdata	2685	2167	2265	2489	2511	2397	2533	19.29	15.64	7.30	6.48	10.73	5.66	2167	0.00	4.52	14.86	15.87	10.61	16.89
la20-sdata	2969	2265	2252	2786	2181	2230	2570	23.71	24.15	6.16	26.54	24.89	13.44	2181	3.85	3.26	27.74	0.00	2.25	17.84
la21-sdata	4219	3265	3606	3767	3227	3691	3142	22.61	14.53	10.71	23.51	12.51	25.53	3142	3.91	14.77	19.89	2.71	17.47	0.00
la22-sdata	3880	3157	3263	3305	3014	3094	3172	18.63	15.90	14.82	22.32	20.26	18.25	3014	4.74	8.26	9.65	0.00	2.65	5.24
la23-sdata	4528	3123	3226	3935	3140	3411	3503	31.03	28.75	13.10	30.65	24.67	22.64	3123	0.00	3.30	26.00	0.54	9.22	12.17
la24-sdata	4231	3224	3106	3610	3337	3159	3637	23.80	26.59	14.68	21.13	25.34	14.04	3106	3.80	0.00	16.23	7.44	1.71	17.10
la25-sdata	4059	3239	3446	3629	3379	3219	3630	20.20	15.10	10.59	16.75	20.69	10.57	3219	0.62	7.05	12.74	4.97	0.00	12.77
la26-sdata	4752	4409	4317	4434	4297	4262	4371	7.22	9.15	6.69	9.57	10.31	8.02	4262	3.45	1.29	4.04	0.82	0.00	2.56
la27-sdata	5600	4717	4351	4656	4431	4293	4739	15.77	22.30	16.86	20.88	23.34	15.38	4293	9.88	1.35	8.46	3.21	0.00	10.39
la28-sdata	5723	4104	4545	4376	4208	4687	4425	28.29	20.58	23.54	26.47	18.10	22.68	4104	0.00	10.75	6.63	2.53	14.21	7.82
la29-sdata	5313	4223	4135	4655	4245	3981	4553	20.52	22.17	12.38	20.10	25.07	14.30	3981	6.08	3.87	16.93	6.63	0.00	14.37
la30-sdata	5247	4465	4456	4577	4681	4392	4645	14.90	15.08	12.77	10.79	16.30	11.47	4392	1.66	1.46	4.21	6.58	0.00	5.76
la31-sdata	7916	6772	6716	6900	6663	6637	6699	14.45	15.16	12.83	15.83	16.16	15.37	6637	2.03	1.19	3.96	0.39	0.00	0.93
la32-sdata	8067	7211	6614	7357	6564	6922	6539	10.61	18.01	8.80	18.63	14.19	18.94	6539	10.28	1.15	12.51	0.38	5.86	0.00
la33-sdata	8417	6776	6402	7181	6608	6581	6608	19.50	23.94	14.68	21.49	21.81	21.49	6402	5.84	0.00	12.17	3.22	2.80	3.22
la34-sdata	8720	6753	6520	7294	6897	6593	6844	22.56	25.23	16.35	20.91	24.39	21.51	6520	3.57	0.00	11.87	5.78	1.12	4.97
la35-sdata	7974	6847	6401	6724	6438	6621	6515	14.13	19.73	15.68	19.26	16.97	18.30	6401	6.97	0.00	5.05	0.58	3.44	1.78
la36-sdata	5708	4518	4042	4681	4078	3975	4232	20.85	29.19	17.99	28.56	30.36	25.86	3975	13.66	1.69	17.76	2.59	0.00	6.47
la37-sdata	5905	4392	4516	4506	4166	4421	4284	25.62	23.52	23.69	29.45	25.13	27.45	4166	5.42	8.40	8.16	0.00	6.12	2.83
la38-sdata	5379	3830	4267	4465	3973	4309	4404	28.80	20.67	16.99	26.14	19.89	18.13	3830	0.00	11.41	16.58	3.73	12.51	14.99
la39-sdata	5901	4174	4424	4631	3989	3866	4509	29.27	25.03	21.52	32.40	34.49	23.59	3866	7.97	14.43	19.79	3.18	0.00	16.63
la40-sdata	5310	4217	4218	4275	4547	4530	4828	20.58	20.56	19.49	14.37	14.69	9.08	4217	0.00	0.02	1.38	7.83	7.42	14.49
Average								16.85	17.93	11.38	17.32	16.52	13.96		4.21	2.92	11.30	3.58	4.63	8.00

$A1 : TSBJT - N^s/N^p$, $A2 : TSBJT - N^c$, $A3 : TSAP - N^p - N^s$, $A4 : TSAP - N^s - N^c$, $A5 : TSAP - N^p - N^c$, and $A6 : TSVN - N^s - N^p - N^c$.

Table 3-19: Performance of the TS algorithms on *sdata* set.

edata	Makespan							Improvement (%)						Best	Deviation from the best (%)					
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)	(17)	(18)	(19)	(20)	(21)
Instance	Initial	A1	A2	A3	A4	A5	A6	A1	A2	A3	A4	A5	A6		A1	A2	A3	A4	A5	A6
la01-edata	1909	1802	1556	1901	1603	1845	1603	5.61	18.49	0.42	16.03	3.35	16.03	1556	15.81	0.00	22.17	3.02	18.57	3.02
la02-edata	1795	1548	1588	1781	1628	1627	1781	13.76	11.53	0.78	9.30	9.36	0.78	1548	0.00	2.58	15.05	5.17	5.10	15.05
la03-edata	1666	1354	1422	1534	1422	1472	1384	18.73	14.65	7.92	14.65	11.64	16.93	1354	0.00	5.02	13.29	5.02	8.71	2.22
la04-edata	1782	1503	1436	1547	1503	1532	1503	15.66	19.42	13.19	15.66	14.03	15.66	1436	4.67	0.00	7.73	4.67	6.69	4.67
la05-edata	1740	1484	1425	1592	1583	1530	1661	14.71	18.10	8.51	9.02	12.07	4.54	1425	4.14	0.00	11.72	11.09	7.37	16.56
la06-edata	2611	2284	2132	2378	2233	2148	2212	12.52	18.35	8.92	14.48	17.73	15.28	2132	7.13	0.00	11.54	4.74	0.75	3.75
la07-edata	2429	2217	2128	2256	2065	2128	2256	8.73	12.39	7.12	14.99	12.39	7.12	2065	7.36	3.05	9.25	0.00	3.05	9.25
la08-edata	2385	2141	2162	2354	2266	2291	2266	10.23	9.35	1.30	4.99	3.94	4.99	2141	0.00	0.98	9.95	5.84	7.01	5.84
la09-edata	2709	2264	2389	2532	2221	2261	2375	16.43	11.81	6.53	18.01	16.54	12.33	2221	1.94	7.56	14.00	0.00	1.80	6.93
la10-edata	2508	2326	2265	2346	2287	2315	2326	7.26	9.69	6.46	8.81	7.70	7.26	2265	2.69	0.00	3.58	0.97	2.21	2.69
la11-edata	3311	3015	3152	3085	2980	3205	3007	8.94	4.80	6.83	10.00	3.20	9.18	2980	1.17	5.77	3.52	0.00	7.55	0.91
la12-edata	3161	2879	2986	3097	2644	2992	2804	8.92	5.54	2.02	16.36	5.35	11.29	2644	8.89	12.93	17.13	0.00	13.16	6.05
la13-edata	3633	3143	2964	3248	3048	2987	3344	13.49	18.41	10.60	16.10	17.78	7.95	2964	6.04	0.00	9.58	2.83	0.78	12.82
la14-edata	3421	3080	3306	3294	3123	3171	3076	9.97	3.36	3.71	8.71	7.31	10.08	3076	0.13	7.48	7.09	1.53	3.09	0.00
la15-edata	3274	2911	3031	2995	2992	3106	2975	11.09	7.42	8.52	8.61	5.13	9.13	2911	0.00	4.12	2.89	2.78	6.70	2.20
la16-edata	2994	2138	2252	2385	2206	2553	2319	28.59	24.78	20.34	26.32	14.73	22.55	2138	0.00	5.33	11.55	3.18	19.41	8.47
la17-edata	2462	1867	1925	2158	1839	2254	2197	24.17	21.81	12.35	25.30	8.45	10.76	1839	1.52	4.68	17.35	0.00	22.57	19.47
la18-edata	3027	2241	2065	2538	2509	2632	2538	25.97	31.78	16.15	17.11	13.05	16.15	2065	8.52	0.00	22.91	21.50	27.46	22.91
la19-edata	3035	1993	1921	2231	2110	2154	2055	34.33	36.71	26.49	30.48	29.03	32.29	1921	3.75	0.00	16.14	9.84	12.13	6.98
la20-edata	2773	2179	2170	2438	2150	2540	2211	21.42	21.75	12.08	22.47	8.40	20.27	2150	1.35	0.93	13.40	0.00	18.14	2.84
la21-edata	4235	3193	3478	3906	2945	3342	3553	24.60	17.87	7.77	30.46	21.09	16.10	2945	8.42	18.10	32.63	0.00	13.48	20.65
la22-edata	4109	3137	3178	3151	3078	3367	3139	23.66	22.66	23.31	25.09	18.06	23.61	3078	1.92	3.25	2.37	0.00	9.39	1.98
la23-edata	4167	3204	2851	3237	2916	3033	3178	23.11	31.58	22.32	30.02	27.21	23.73	2851	12.38	0.00	13.54	2.28	6.38	11.47
la24-edata	4416	3059	2928	3728	3364	3238	3367	30.73	33.70	15.58	23.82	26.68	23.75	2928	4.47	0.00	27.32	14.89	10.59	14.99
la25-edata	4182	3239	3387	3427	3283	3018	3336	22.55	19.01	18.05	21.50	27.83	20.23	3018	7.32	12.23	13.55	8.78	0.00	10.54
la26-edata	5270	3963	4155	4078	4112	4301	4006	24.80	21.16	22.62	21.97	18.39	23.98	3963	0.00	4.84	2.90	3.76	8.53	1.09
la27-edata	5422	4534	4217	4713	4379	4518	4447	16.38	22.22	13.08	19.24	16.67	17.98	4217	7.52	0.00	11.76	3.84	7.14	5.45
la28-edata	5094	4113	4435	4397	4408	4328	4119	19.26	12.94	13.68	13.47	15.04	19.14	4113	0.00	7.83	6.90	7.17	5.23	0.15
la29-edata	5083	4189	4164	4462	4133	4219	4278	17.59	18.08	12.22	18.69	17.00	15.84	4133	1.35	0.75	7.96	0.00	2.08	3.51
la30-edata	5611	4176	4210	4484	4177	4294	4153	25.57	24.97	20.09	25.56	23.47	25.98	4153	0.55	1.37	7.97	0.58	3.40	0.00
la31-edata	8258	6189	5912	6175	5479	5778	6039	25.05	28.41	25.22	33.65	30.03	26.87	5479	12.96	7.90	12.70	0.00	5.46	10.22
la32-edata	9129	7195	6403	7265	6349	6262	6599	21.19	29.86	20.42	30.45	31.41	27.71	6262	14.90	2.25	16.02	1.39	0.00	5.38
la33-edata	8444	6464	6383	6527	6636	5933	6503	23.45	24.41	22.70	21.41	29.74	22.99	5933	8.95	7.58	10.01	11.85	0.00	9.61
la34-edata	8051	6670	6390	6804	6634	6104	6509	17.15	20.63	15.49	17.60	24.18	19.15	6104	9.27	4.69	11.47	8.68	0.00	6.63
la35-edata	7905	6895	6228	7452	6155	6087	6117	12.78	21.21	5.73	22.14	23.00	22.62	6087	13.27	2.32	22.42	1.12	0.00	0.49
la36-edata	5708	3938	4027	4139	3984	3721	4361	31.01	29.45	27.49	30.20	34.81	23.60	3721	5.83	8.22	11.23	7.07	0.00	17.20
la37-edata	5869	4572	4174	4441	4417	4589	4627	22.10	28.88	24.33	24.74	21.81	21.16	4174	9.54	0.00	6.40	5.82	9.94	10.85
la38-edata	5978	4185	3826	4236	3933	3775	4079	29.99	36.00	29.14	34.21	36.85	31.77	3775	10.86	1.35	12.21	4.19	0.00	8.05
la39-edata	5818	3927	3906	4261	3451	3871	4413	32.50	32.86	26.76	40.68	33.47	24.15	3451	13.79	13.18	23.47	0.00	12.17	27.88
la40-edata	5896	4321	4092	3880	4394	3803	3932	26.71	30.60	34.19	25.47	35.50	33.31	3803	13.62	7.60	2.02	15.54	0.00	3.39
Average								19.52	20.67	14.51	20.44	18.34	17.86		5.80	4.10	12.37	4.48	7.15	8.05

$A1 : TSBJT - N^s/N^p$, $A2 : TSBJT - N^c$, $A3 : TSAP - N^p - N^s$, $A4 : TSAP - N^s - N^c$, $A5 : TSAP - N^p - N^c$, and $A6 : TSVN - N^s - N^p - N^c$.

Table 3-20: Performance of the TS algorithms on *edata* set.

rdata	Makespan							Improvement (%)						Best	Deviation from the best (%)					
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)	(17)	(18)	(19)	(20)	(21)
Instance	Initial	A1	A2	A3	A4	A5	A6	A1	A2	A3	A4	A5	A6		A1	A2	A3	A4	A5	A6
la01-rdata	1974	1298	1341	1482	1490	1635	1434	34.25	32.07	24.92	24.52	17.17	27.36	1298	0.00	3.31	14.18	14.79	25.96	10.48
la02-rdata	1887	1420	1460	1539	1462	1677	1462	24.75	22.63	18.44	22.52	11.13	22.52	1420	0.00	2.82	8.38	2.96	18.10	2.96
la03-rdata	1594	1178	1235	1298	1170	1411	1237	26.10	22.52	18.57	26.60	11.48	22.40	1170	0.68	5.56	10.94	0.00	20.60	5.73
la04-rdata	1685	1235	1216	1247	1237	1269	1247	26.71	27.83	25.99	26.59	24.69	25.99	1216	1.56	0.00	2.55	1.73	4.36	2.55
la05-rdata	1664	1128	1147	1212	1136	1256	1148	32.21	31.07	27.16	31.73	24.52	31.01	1128	0.00	1.68	7.45	0.71	11.35	1.77
la06-rdata	2570	1978	2213	2359	2086	2242	2280	23.04	13.89	8.21	18.83	12.76	11.28	1978	0.00	11.88	19.26	5.46	13.35	15.27
la07-rdata	2406	1868	1902	2097	1897	1984	2061	22.36	20.95	12.84	21.16	17.54	14.34	1868	0.00	1.82	12.26	1.55	6.21	10.33
la08-rdata	2530	1847	2066	2044	2044	2105	2040	27.00	18.34	19.21	19.21	16.80	19.37	1847	0.00	11.86	10.67	10.67	13.97	10.45
la09-rdata	2757	1963	1996	2032	1959	2100	2169	28.80	27.60	26.30	28.94	23.83	21.33	1959	0.20	1.89	3.73	0.00	7.20	10.72
la10-rdata	2781	1937	2050	2149	2153	2219	2037	30.35	26.29	22.73	22.58	20.21	26.75	1937	0.00	5.83	10.94	11.15	14.56	5.16
la11-rdata	3386	2702	2912	3129	2894	2972	2757	20.20	14.00	7.59	14.53	12.23	18.58	2702	0.00	7.77	15.80	7.11	9.99	2.04
la12-rdata	3033	2290	2493	2386	2391	2639	2396	24.50	17.80	21.33	21.17	12.99	21.00	2290	0.00	8.86	4.19	4.41	15.24	4.63
la13-rdata	3232	2598	2588	2852	2495	2502	2703	19.62	19.93	11.76	22.80	22.59	16.37	2495	4.13	3.73	14.31	0.00	0.28	8.34
la14-rdata	3094	2470	2605	2443	2463	2741	2444	20.17	15.80	21.04	20.39	11.41	21.01	2443	1.11	6.63	0.00	0.82	12.20	0.04
la15-rdata	3329	2583	2766	2736	2771	2806	2724	22.41	16.91	17.81	16.76	15.71	18.17	2583	0.00	7.08	5.92	7.28	8.63	5.46
la16-rdata	2907	1550	1744	1910	1947	2143	1937	46.68	40.01	34.30	33.02	26.28	33.37	1550	0.00	12.52	23.23	25.61	38.26	24.97
la17-rdata	2224	1464	1527	1599	1489	1833	1718	34.17	31.34	28.10	33.05	17.58	22.75	1464	0.00	4.30	9.22	1.71	25.20	17.35
la18-rdata	2660	1679	1984	2102	1841	2199	1844	36.88	25.41	20.98	30.79	17.33	30.68	1679	0.00	18.17	25.19	9.65	30.97	9.83
la19-rdata	2879	1655	1832	1938	1810	2034	1672	42.51	36.37	32.68	37.13	29.35	41.92	1655	0.00	10.69	17.10	9.37	22.90	1.03
la20-rdata	2911	1709	1902	2041	2142	2157	2017	41.29	34.66	29.89	26.42	25.90	30.71	1709	0.00	11.29	19.43	25.34	26.21	18.02
la21-rdata	4394	2609	2613	2781	2379	2706	2935	40.62	40.53	36.71	45.86	38.42	33.20	2379	9.67	9.84	16.90	0.00	13.75	23.37
la22-rdata	3582	2409	2527	2725	2567	2689	2640	32.75	29.45	23.93	28.34	24.93	26.30	2409	0.00	4.90	13.12	6.56	11.62	9.59
la23-rdata	4135	2790	2972	2929	2817	3532	2740	32.53	28.13	29.17	31.87	14.58	33.74	2740	1.82	8.47	6.90	2.81	28.91	0.00
la24-rdata	3737	2360	2606	2747	2561	2823	2659	36.85	30.26	26.49	31.47	24.46	28.85	2360	0.00	10.42	16.40	8.52	19.62	12.67
la25-rdata	3461	2363	2613	2392	2516	2690	2392	31.72	24.50	30.89	27.30	22.28	30.89	2363	0.00	10.58	1.23	6.47	13.84	1.23
la26-rdata	5243	3845	3862	3890	4057	3679	4069	26.66	26.34	25.81	22.62	29.83	22.39	3679	4.51	4.97	5.74	10.27	0.00	10.60
la27-rdata	5221	3493	3680	3418	3981	3630	3722	33.10	29.52	34.53	23.75	30.47	28.71	3418	2.19	7.67	0.00	16.47	6.20	8.89
la28-rdata	4977	3517	3655	3784	3529	3880	3937	29.33	26.56	23.97	29.09	22.04	20.90	3517	0.00	3.92	7.59	0.34	10.32	11.94
la29-rdata	5130	3543	3830	3680	3458	3746	3726	30.94	25.34	28.27	32.59	26.98	27.37	3458	2.46	10.76	6.42	0.00	8.33	7.75
la30-rdata	5583	3539	3539	4534	3778	3480	4129	36.61	36.61	18.79	32.33	37.67	26.04	3480	1.70	1.70	30.29	8.56	0.00	18.65
la31-rdata	7548	5404	5560	5442	5654	5446	4883	28.40	26.34	27.90	25.09	27.85	35.31	4883	10.67	13.86	11.45	15.79	11.53	0.00
la32-rdata	8267	5899	5646	5786	6114	6058	5527	28.64	31.70	30.01	26.04	26.72	33.14	5527	6.73	2.15	4.69	10.62	9.61	0.00
la33-rdata	7523	5294	5671	5247	5272	5427	5030	29.63	24.62	30.25	29.92	27.86	33.14	5030	5.25	12.74	4.31	4.81	7.89	0.00
la34-rdata	8194	5412	5770	5108	5682	5736	5272	33.95	29.58	37.66	30.66	30.00	35.66	5108	5.95	12.96	0.00	11.24	12.29	3.21
la35-rdata	7775	5000	6014	5257	6085	5851	5163	35.69	22.65	32.39	21.74	24.75	33.59	5000	0.00	20.28	5.14	21.70	17.02	3.26
la36-rdata	5456	3279	3286	3114	2937	3631	3485	39.90	39.77	42.93	46.17	33.45	36.13	2937	11.64	11.88	6.03	0.00	23.63	18.66
la37-rdata	5971	3491	3505	3471	3473	3817	3668	41.53	41.30	41.87	41.84	36.07	38.57	3471	0.58	0.98	0.00	0.06	9.97	5.68
la38-rdata	5855	2831	2999	3276	3140	3311	3323	51.65	48.78	44.05	46.37	43.45	43.25	2831	0.00	5.93	15.72	10.91	16.96	17.38
la39-rdata	5206	2981	2945	3368	3151	3369	3214	42.74	43.43	35.31	39.47	35.29	38.26	2945	1.22	0.00	14.36	6.99	14.40	9.13
la40-rdata	5528	3338	3471	3223	3389	3698	3336	39.62	37.21	41.70	38.69	33.10	39.65	3223	3.57	7.69	0.00	5.15	14.74	3.51
Average								32.17	28.45	26.81	28.75	24.04	28.05		1.89	7.49	10.03	7.19	14.40	8.32

$A1 : TSBJT - N^s/N^p$, $A2 : TSBJT - N^c$, $A3 : TSAP - N^p - N^s$, $A4 : TSAP - N^s - N^c$, $A5 : TSAP - N^p - N^c$, and $A6 : TSVN - N^s - N^p - N^c$.

Table 3-21: Performance of the TS algorithms on *rdata* set.

vdata	Makespan							Improvement (%)						Best	Deviation from the best (%)					
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)	(17)	(18)	(19)	(20)	(21)
Instance	Initial	A1	A2	A3	A4	A5	A6	A1	A2	A3	A4	A5	A6		A1	A2	A3	A4	A5	A6
la01-vdata	1829	1281	1336	1578	1450	1527	1533	29.96	26.95	13.72	20.72	16.51	16.18	1281	0.00	4.29	23.19	13.19	19.20	19.67
la02-vdata	1685	1172	1272	1479	1301	1337	1364	30.45	24.51	12.23	22.79	20.65	19.05	1172	0.00	8.53	26.19	11.01	14.08	16.38
la03-vdata	1609	1099	1169	1270	1266	1290	1272	31.70	27.35	21.07	21.32	19.83	20.94	1099	0.00	6.37	15.56	15.20	17.38	15.74
la04-vdata	1455	1165	1110	1214	1163	1398	1204	19.93	23.71	16.56	20.07	3.92	17.25	1110	4.95	0.00	9.37	4.77	25.95	8.47
la05-vdata	1639	1148	1206	1289	1186	1426	1344	29.96	26.42	21.35	27.64	13.00	18.00	1148	0.00	5.05	12.28	3.31	24.22	17.07
la06-vdata	2503	1730	1803	1855	1859	2004	2034	30.88	27.97	25.89	25.73	19.94	18.74	1730	0.00	4.22	7.23	7.46	15.84	17.57
la07-vdata	2561	1709	1883	2008	1678	1872	1987	33.27	26.47	21.59	34.48	26.90	22.41	1678	1.85	12.22	19.67	0.00	11.56	18.41
la08-vdata	2491	1721	1816	1807	1784	1912	1833	30.91	27.10	27.46	28.38	23.24	26.42	1721	0.00	5.52	5.00	3.66	11.10	6.51
la09-vdata	2632	1812	1908	2112	1986	2128	1959	31.16	27.51	19.76	24.54	19.15	25.57	1812	0.00	5.30	16.56	9.60	17.44	8.11
la10-vdata	2668	1781	1938	2028	1955	2053	2040	33.25	27.36	23.99	26.72	23.05	23.54	1781	0.00	8.82	13.87	9.77	15.27	14.54
la11-vdata	3160	2411	2705	2477	2545	2506	2513	23.70	14.40	21.61	19.46	20.70	20.47	2411	0.00	12.19	2.74	5.56	3.94	4.23
la12-vdata	2985	2268	2352	2403	2214	2343	2384	24.02	21.21	19.50	25.83	21.51	20.13	2214	2.44	6.23	8.54	0.00	5.83	7.68
la13-vdata	3116	2355	2523	2712	2406	2441	2364	24.42	19.03	12.97	22.79	21.66	24.13	2355	0.00	7.13	15.16	2.17	3.65	0.38
la14-vdata	3383	2452	2607	2521	2307	2584	2567	27.52	22.94	25.48	31.81	23.62	24.12	2307	6.29	13.00	9.28	0.00	12.01	11.27
la15-vdata	3133	2349	2624	2586	2426	2772	2742	25.02	16.25	17.46	22.57	11.52	12.48	2349	0.00	11.71	10.09	3.28	18.01	16.73
la16-vdata	2585	1425	1383	1510	1362	1624	1482	44.87	46.50	41.59	47.31	37.18	42.67	1362	4.63	1.54	10.87	0.00	19.24	8.81
la17-vdata	2378	1262	1222	1316	1348	1434	1394	46.93	48.61	44.66	43.31	39.70	41.38	1222	3.27	0.00	7.69	10.31	17.35	14.08
la18-vdata	2556	1266	1453	1475	1408	1794	1555	50.47	43.15	42.29	44.91	29.81	39.16	1266	0.00	14.77	16.51	11.22	41.71	22.83
la19-vdata	2556	1335	1577	1520	1348	1545	1445	47.77	38.30	40.53	47.26	39.55	43.47	1335	0.00	18.13	13.86	0.97	15.73	8.24
la20-vdata	2885	1410	1549	1573	1396	1684	1720	51.13	46.31	45.48	51.61	41.63	40.38	1396	1.00	10.96	12.68	0.00	20.63	23.21
la21-vdata	3525	2156	2010	2069	2020	2458	2176	38.84	42.98	41.30	42.70	30.27	38.27	2010	7.26	0.00	2.94	0.50	22.29	8.26
la22-vdata	3664	1978	2080	2080	1881	2027	1884	46.02	43.23	43.23	48.66	44.68	48.58	1881	5.16	10.58	10.58	0.00	7.76	0.16
la23-vdata	3641	2048	2174	2016	1979	2205	2054	43.75	40.29	44.63	45.65	39.44	43.59	1979	3.49	9.85	1.87	0.00	11.42	3.79
la24-vdata	3711	1936	2080	1823	1882	2344	1887	47.83	43.95	50.88	49.29	36.84	49.15	1823	6.20	14.10	0.00	3.24	28.58	3.51
la25-vdata	3610	2050	2237	2238	1922	2477	2143	43.21	38.03	38.01	46.76	31.39	40.64	1922	6.66	16.39	16.44	0.00	28.88	11.50
la26-vdata	5032	2872	3117	2699	3561	3167	2652	42.93	38.06	46.36	29.23	37.06	47.30	2652	8.30	17.53	1.77	34.28	19.42	0.00
la27-vdata	4842	2710	3007	2871	3561	3029	2722	44.03	37.90	40.71	26.46	37.44	43.78	2710	0.00	10.96	5.94	31.40	11.77	0.44
la28-vdata	5038	2718	3085	2682	3537	3120	2506	46.05	38.77	46.76	29.79	38.07	50.26	2506	8.46	23.10	7.02	41.14	24.50	0.00
la29-vdata	4774	2715	2817	2486	3315	3047	2598	43.13	40.99	47.93	30.56	36.18	45.58	2486	9.21	13.31	0.00	33.35	22.57	4.51
la30-vdata	4591	2664	2810	2554	3667	2835	2365	41.97	38.79	44.37	20.13	38.25	48.49	2365	12.64	18.82	7.99	55.05	19.87	0.00
la31-vdata	6784	4611	5021	3780	6223	4959	3851	32.03	25.99	44.28	8.27	26.90	43.23	3780	21.98	32.83	0.00	64.63	31.19	1.88
la32-vdata	8327	5622	5843	4088	7563	6048	4926	32.48	29.83	50.91	9.17	27.37	40.84	4088	37.52	42.93	0.00	85.00	47.95	20.50
la33-vdata	7244	5080	5470	3839	6421	5533	4540	29.87	24.49	47.00	11.36	23.62	37.33	3839	32.33	42.49	0.00	67.26	44.13	18.26
la34-vdata	6612	4747	5243	3586	6148	5187	3849	28.21	20.70	45.77	7.02	21.55	41.79	3586	32.38	46.21	0.00	71.44	44.65	7.33
la35-vdata	6194	4447	4974	3912	5644	5006	4005	28.20	19.70	36.84	8.88	19.18	35.34	3912	13.68	27.15	0.00	44.27	27.97	2.38
la36-vdata	5640	3294	2677	2332	4295	2738	2230	41.60	52.54	58.65	23.85	51.45	60.46	2230	47.71	20.04	4.57	92.60	22.78	0.00
la37-vdata	4926	2857	2884	2584	3899	3140	2402	42.00	41.45	47.54	20.85	36.26	51.24	2402	18.94	20.07	7.58	62.32	30.72	0.00
la38-vdata	4219	2585	2424	2418	3526	2618	2174	38.73	42.55	42.69	16.43	37.95	48.47	2174	18.91	11.50	11.22	62.19	20.42	0.00
la39-vdata	5478	2606	2672	2146	4288	2681	2268	52.43	51.22	60.83	21.72	51.06	58.60	2146	21.44	24.51	0.00	99.81	24.93	5.68
la40-vdata	5225	2533	2421	2186	4259	2830	2178	51.52	53.67	58.16	18.49	45.84	58.32	2178	16.30	11.16	0.37	95.55	29.94	0.00
Average								37.05	33.68	36.30	28.11	29.60	36.19		8.82	14.24	8.37	26.39	21.30	8.70

$A1 : TSBJT - N^s/N^p$, $A2 : TSBJT - N^c$, $A3 : TSAP - N^p - N^s$, $A4 : TSAP - N^s - N^c$, $A5 : TSAP - N^p - N^c$, and $A6 : TSVN - N^s - N^p - N^c$.

Table 3-22: Performance of the TS algorithms on *vdata* set.

We also evaluated the performance of the three generic TS algorithms based on their derived TS algorithms. The results are summarized in tables 3-23 and 3-24. Observe from these two tables that:

1. In general, the generic *TSBJT* performed best, *TSVN* second best, and *TSAP* worst;
2. Increasing flexibility improved average improvement over the initial solutions but did not always improve the other metrics.

	Generic TS algorithms			Rank		
	TSBJT	*TSAP*	*TSVN*	*TSBJT*	*TSAP*	*TSVN*
Overall						
Avg. Imp.(%)	25.79	22.68	24.02	*1*	*3*	*2*
♯ best Cmax	83	67	16	*1*	*2*	*3*
Avg dev. (%)	6.18	10.55	7.93	*1*	*3*	*2*
Avg. ranks				*1.00*	*2.67*	*2.33*
sdata						
Avg. Imp.(%)	17.39	15.07	13.96	*1*	*2*	*3*
♯ best Cmax	24	19	3	*1*	*2*	*3*
Avg dev. (%)	3.57	6.50	8.00	*1*	*2*	*3*
Avg. ranks				*1.00*	*2.00*	*3.00*
edata						
Avg. Imp.(%)	20.09	17.76	17.86	*1*	*3*	*2*
♯ best Cmax	19	19	2	*1*	*1*	*3*
Avg dev. (%)	4.95	8.00	8.05	*1*	*2*	*3*
Avg. ranks				*1.00*	*2.00*	*2.67*
rdata						
Avg. Imp.(%)	30.31	26.53	28.05	*1*	*3*	*2*
♯ best Cmax	23	13	4	*1*	*2*	*3*
Avg dev. (%)	4.69	10.54	8.32	*1*	*3*	*2*
Avg. ranks				*1.00*	*2.67*	*2.33*
vdata						
Avg. Imp.(%)	35.37	31.34	36.19	*2*	*3*	*1*
♯ best Cmax	17	16	7	*1*	*2*	*3*
Avg dev. (%)	11.53	18.68	8.70	*2*	*3*	*1*
Avg. ranks				*1.67*	*2.67*	*1.67*

Avg. Imp.(%) - average improvement (in %)
♯ best Cmax - number of best makespans obtained
Avg dev. (%) - average deviation from the best makespan (in %)
Avg. ranks - average ranks of the generic TS algorithms

Table 3-23: Ranks of the generic TS algorithms w.r.t. flexibility levels.

Finally, we assessed how the TS algorithms' performance depends on different initial solutions. The influence of an initial solution on a performance of a local search is an open question, which depends on a problem under study and an employed algorithm. Computational experiments on the performance of Nowicki and Smutnicki's tabu search for the JS showed that a better initial solution is likely to lead to a better final result [JRM00]. On the other hand, tabu search algorithms for extended job shop problems seem to be less influenced by the starting solutions, e.g. in [DPRL98]. In our experiment, to reduce the computing time to run all 160 instances, for each data set we randomly picked up eight instances, one per each subgroup of

the same instance size. So there were 32 randomly selected instances in total. We run all TS algorithms on each instance with three initial solutions obtained by three constructive heuristics *SG-MCO*, *SG-MFP*, and *SG-BN*. Then for each algorithm, we compare its performance with respect to different constructive heuristics, i.e.we compare *SG-MCO* versus *SG-MFP*, *SG-MCO* versus *SG-BN*, and *SG-MFP* versus *SG-BN*. We defined that an instance is *nonconformity* with respect to a TS algorithm and two initial solution generators if one initial solution generating method performs better than another but results in an inferior makespan when being used in the TS algorithm. Obviously, if a TS's performance were dependent on initial solutions' quality then its number of nonconformity instance would be zero. Nonconformity for each pair of different initial schedule generators are reported in Tables 3-25 to 3-27. These tables show that no TS algorithm is dependent on initial solutions' quality to yield good results. This finding suggests that we can also use permutation schedules as initial solutions for the TS algorithms. This was confirmed by running the TS algorithms with five initial permutation schedules and comparing the best makespan obtained with random permutations with the best makespans obtained with constructive heuristics. Although constructive heuristics always outperformed permutation schedule procedures, their associated makespans by the TS algorithms were inferior in many cases (see Table 3-28).

Instance	Generic TS algorithms			Rank		
subgroup	TSBJT	TSAP	TSVN	TSBJT	TSAP	TSVN
la01-la05 (10x5)						
Avg. Imp.(%)	21.43	15.70	16.52	*1*	*3*	*2*
♯ best Cmax	20	4	1	*1*	*2*	*3*
Avg dev. (%)	2.55	10.28	9.16	*1*	*3*	*2*
Avg. ranks				*1.00*	*2.67*	*2.33*
la06-la10 (15x5)						
Avg. Imp.(%)	18.81	15.28	14.83	*1*	*2*	*3*
♯ best Cmax	15	5	0	*1*	*2*	*3*
Avg dev. (%)	3.10	7.76	8.42	*1*	*2*	*3*
Avg. ranks				*1.00*	*2.00*	*3.00*
la11-la15 (20x5)						
Avg. Imp.(%)	15.03	13.93	14.04	*1*	*3*	*2*
♯ best Cmax	10	10	2	*1*	*1*	*3*
Avg dev. (%)	4.56	5.91	5.83	*1*	*3*	*2*
Avg. ranks				*1.00*	*2.33*	*2.33*
la16-la20 (10x10)						
Avg. Imp.(%)	32.98	26.20	26.60	*1*	*3*	*2*
♯ best Cmax	15	5	4	*1*	*2*	*3*
Avg dev. (%)	3.95	14.57	13.84	*1*	*3*	*2*
Avg. ranks				*1.00*	*2.67*	*2.33*
la21-la25 (15x10)						
Avg. Imp.(%)	30.55	28.18	28.58	*1*	*3*	*2*
♯ best Cmax	8	10	4	*2*	*1*	*3*
Avg dev. (%)	6.25	9.79	9.05	*1*	*3*	*2*
Avg. ranks				*1.33*	*2.33*	*2.33*
la26-la30 (20x10)						
Avg. Imp.(%)	27.31	24.87	26.78	*1*	*3*	*2*
♯ best Cmax	6	10	4	*2*	*1*	*3*
Avg dev. (%)	5.66	9.67	5.69	*1*	*3*	*2*
Avg. ranks				*1.33*	*2.33*	*2.33*
la31-la35 (30x10)						
Avg. Imp.(%)	24.25	23.80	29.22	*2*	*3*	*1*
♯ best Cmax	4	12	0	*2*	*1*	*3*
Avg dev. (%)	13.38	14.11	5.00	*2*	*3*	*1*
Avg. ranks				*2.00*	*2.33*	*1.67*
la36-la40 (15x15)						
Avg. Imp.(%)	35.95	33.44	35.55	*1*	*3*	*2*
♯ best Cmax	5	11	0	*2*	*1*	*3*
Avg dev. (%)	10.03	15.35	9.14	*2*	*3*	*1*
Avg. ranks				*1.67*	*2.33*	*2.00*
Avg. Imp.(%) - average improvement (in %)						
♯ best Cmax - number of best makespans obtained						
Avg dev. (%) - average deviation from the best makespan (in %)						
Avg. ranks - average ranks of the generic TS algorithms						

Table 3-24: Ranks of the generic Tabu search algorithms w.r.t. instance sizes.

		Makespans with SG-MFP							Makespans with SG-BN						Conformity					
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)	(17)	(18)	(19)	(20)	(21)
Instance	SG-MFP	A1	A2	A3	A4	A5	A6	SG-BN	A1	A2	A3	A4	A5	A6	A1	A2	A3	A4	A5	A6
la01-sdata	1988	1773	1583	1773	1631	1816	1773	1737	1737	1588	1737	1677	1677	1677	1	0	1	0	1	1
la06-sdata	2472	2333	2229	2369	2342	2360	2342	2577	2280	2254	2500	2350	2254	2350	0	1	1	1	0	1
la11-sdata	3526	3142	3065	3318	2961	2961	3144	3653	3329	3315	3522	3092	3202	3518	1	1	1	1	1	1
la16-sdata	2886	2303	2146	2351	2157	2367	2448	3091	2347	2137	2410	2387	2407	2412	1	0	1	1	1	0
la21-sdata	4219	3265	3606	3767	3227	3691	3142	4720	3608	3368	3548	3162	3379	3432	1	0	0	0	0	1
la26-sdata	4752	4409	4317	4434	4297	4262	4371	5996	4413	4087	5007	4207	4430	4201	1	0	1	0	1	0
la31-sdata	7916	6772	6716	6900	6663	6637	6699	8654	6914	6596	6802	6736	6208	6593	1	0	0	1	0	0
la36-sdata	5708	4518	4042	4681	4078	3975	4232	6587	4504	4657	4710	3982	4145	4270	0	1	1	0	1	1
Number of nonconformity instances									2	5	2	4	3	3	6	3	6	4	5	5
la01-edata	1909	1802	1556	1901	1603	1845	1603	2320	1582	1545	1676	1550	1644	1653	0	0	0	0	0	1
la06-edata	2611	2284	2132	2378	2233	2148	2212	2605	2291	2283	2297	2096	2365	2297	0	0	1	1	0	0
la11-edata	3311	3015	3152	3085	2980	3205	3007	3586	2941	2957	3033	2839	3111	3161	0	0	0	0	0	1
la16-edata	2994	2138	2252	2385	2206	2553	2319	3392	2155	2232	2416	1804	2593	2366	1	0	1	0	1	1
la21-edata	4235	3193	3478	3906	2945	3342	3553	4630	3269	3411	3446	3223	3348	3351	1	0	0	1	1	0
la26-edata	5270	3963	4155	4078	4112	4301	4006	5544	4195	4299	4213	3318	4104	4320	1	1	1	0	0	1
la31-edata	8258	6189	5912	6175	5479	5778	6039	7982	6412	5791	6216	6133	5855	6285	0	1	0	0	0	0
la36-edata	5708	3938	4027	4139	3984	3721	4361	6373	4079	4176	4502	3105	3904	4044	1	1	1	0	1	0
Number of nonconformity instances									4	5	4	6	5	4	4	3	4	2	3	4
la01-rdata	1974	1298	1341	1482	1490	1635	1434	2206	1441	1438	1527	1550	1603	1482	1	1	1	1	0	1
la06-rdata	2570	1978	2213	2359	2086	2242	2280	2647	1942	2101	2242	2096	2085	2207	0	0	0	1	0	0
la11-rdata	3386	2702	2912	3129	2894	2972	2757	3296	2843	2908	3024	2839	2905	2987	0	1	1	1	1	0
la16-rdata	2907	1550	1744	1910	1947	2143	1937	3154	1551	1706	2016	1804	2047	1906	1	0	1	0	0	0
la21-rdata	4394	2609	2613	2781	2379	2706	2935	4557	2709	2657	3025	3223	2749	2927	1	1	1	1	1	0
la26-rdata	5243	3845	3862	3890	4057	3679	4069	5306	3488	3332	3527	3318	3627	3581	0	0	0	0	0	0
la31-rdata	7548	5404	5560	5442	5654	5446	4883	8213	6336	5799	5372	6133	6183	6197	1	1	0	1	1	1
la36-rdata	5456	3279	3286	3114	2937	3631	3485	5844	3007	2975	3296	3105	3541	3542	0	0	1	1	0	1
Number of nonconformity instances									4	4	3	2	5	5	4	4	5	6	3	3
la01-vdata	1829	1281	1336	1578	1450	1527	1533	2004	1258	1272	1423	1328	1445	1351	0	0	0	0	0	0
la06-vdata	2503	1730	1803	1855	1859	2004	2034	2594	1816	1898	2113	1793	2087	2055	1	1	1	0	1	1
la11-vdata	3160	2411	2705	2477	2545	2506	2513	3319	2529	2720	2646	2559	2634	2647	1	1	1	1	1	1
la16-vdata	2585	1425	1383	1510	1362	1624	1482	2568	1281	1394	1453	1382	1706	1435	1	0	1	0	0	1
la21-vdata	3525	2156	2010	2069	2020	2458	2176	4474	2122	2399	1886	2099	2443	2074	0	1	0	1	0	0
la26-vdata	5032	2872	3117	2699	3561	3167	2652	4991	2709	2693	2805	3603	3077	2678	1	1	0	0	1	0
la31-vdata	6784	4611	5021	3780	6223	4959	3851	7753	5279	5498	4208	7359	5448	4092	1	1	1	1	1	1
la36-vdata	5640	3294	2677	2332	4295	2738	2230	6017	2861	2918	2372	4580	2785	2759	0	1	1	1	1	1
Number of nonconformity instances									3	2	3	4	3	3	5	6	5	4	5	5

Table 3-25: Nonconformity with SG-MFP and SG-MCO as initial schedule generators.

	Makespans with SG-MFP						Makespans with SG-BN						Conformity					
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)	(17)	(18)	(19)	(20)	(21)
Instance	SG-MFP	A1	A2	A3	A4	A5	A6	SG-BN	A1	A2	A3	A4	A5	A6	A1	A2	A3	A4	A5	A6
la01-sdata	1988	1773	1583	1773	1631	1816	1773	2368	2083	1578	2117	1798	1703	1639	1	0	1	1	0	0
la06-sdata	2472	2333	2229	2369	2342	2360	2342	3004	2384	2326	2834	2257	2374	2426	1	1	1	0	1	1
la11-sdata	3526	3142	3065	3318	2961	2961	3144	3461	3200	3082	3229	2938	3003	3205	0	0	1	1	0	0
la16-sdata	2886	2303	2146	2351	2157	2367	2448	3779	2312	2307	2743	2017	2296	2501	1	1	1	0	0	1
la21-sdata	4219	3265	3606	3767	3227	3691	3142	4975	3600	3615	4096	3685	3665	3747	1	1	1	1	0	1
la26-sdata	4752	4409	4317	4434	4297	4262	4371	6119	4601	4351	5248	4767	4399	4713	1	1	1	1	1	1
la31-sdata	7916	6772	6716	6900	6663	6637	6699	9155	7601	6226	8157	6329	6293	6163	1	0	1	0	0	0
la36-sdata	5708	4518	4042	4681	4078	3975	4232	7714	4865	4328	5357	4367	4409	5133	1	1	1	1	1	1
Number of nonconformity instances									1	3	0	3	5	3	7	5	8	5	3	5
la01-edata	1909	1802	1556	1901	1603	1845	1603	2192	1631	1564	1797	1624	1685	1643	0	1	0	1	0	1
la06-edata	2611	2284	2132	2378	2233	2148	2212	3189	2295	2305	2789	2210	2239	2415	1	1	1	0	1	1
la11-edata	3311	3015	3152	3085	2980	3205	3007	3813	2952	3029	3131	3068	3022	2964	0	0	1	1	0	0
la16-edata	2994	2138	2252	2385	2206	2553	2319	3847	2001	1988	2408	2399	2420	2198	0	0	1	1	0	0
la21-edata	4235	3193	3478	3906	2945	3342	3553	5036	3820	3247	3753	3506	3480	3342	1	0	0	1	1	0
la26-edata	5270	3963	4155	4078	4112	4301	4006	6365	4711	4301	5063	4687	4521	4476	1	1	1	1	1	1
la31-edata	8258	6189	5912	6175	5479	5778	6039	9788	7435	5967	7575	6326	6042	6301	1	1	1	1	1	1
la36-edata	5708	3938	4027	4139	3984	3721	4361	8030	4557	4219	5994	4129	4247	4485	1	1	1	1	1	1
Number of nonconformity instances									3	3	2	1	3	3	5	5	6	7	5	5
la01-rdata	1974	1298	1341	1482	1490	1635	1434	2039	1426	1472	1681	1595	1588	1508	1	1	1	1	0	1
la06-rdata	2570	1978	2213	2359	2086	2242	2280	2812	2047	2168	2482	2049	2236	2175	1	0	1	0	0	0
la11-rdata	3386	2702	2912	3129	2894	2972	2757	3707	2734	2846	3038	2787	3068	2794	1	0	0	0	1	1
la16-rdata	2907	1550	1744	1910	1947	2143	1937	3398	1631	1728	1881	1880	2076	1868	1	1	1	0	0	0
la21-rdata	4394	2609	2613	2781	2379	2706	2935	4782	2852	2699	3028	2877	2610	2852	1	0	1	0	0	0
la26-rdata	5243	3845	3862	3890	4057	3679	4069	5806	3765	3175	3901	3134	3170	3760	0	0	1	0	0	0
la31-rdata	7548	5404	5560	5442	5654	5446	4883	9015	5694	6521	5660	6600	5453	5162	1	1	1	1	1	1
la36-rdata	5456	3279	3286	3114	2937	3631	3485	7833	3773	3497	4652	3582	3346	3584	1	1	1	1	0	1
Number of nonconformity instances									1	4	2	4	6	4	7	4	6	4	2	4
la01-vdata	1829	1281	1336	1578	1450	1527	1533	1867	1339	1353	1430	1360	1574	1431	1	1	0	0	1	0
la06-vdata	2503	1730	1803	1855	1859	2004	2034	2728	1828	2112	2062	1783	1937	1973	1	1	1	0	0	0
la11-vdata	3160	2411	2705	2477	2545	2506	2513	3261	2502	2648	2636	2684	2711	2637	1	0	1	1	1	1
la16-vdata	2585	1425	1383	1510	1362	1624	1482	2861	1387	1456	1579	1343	1592	1547	0	1	1	0	1	1
la21-vdata	3525	2156	2010	2069	2020	2458	2176	4593	2025	2149	2128	2818	2336	2075	0	1	1	1	0	1
la26-vdata	5032	2872	3117	2699	3561	3167	2652	5842	3181	3083	2497	4658	3203	2545	1	0	0	1	1	0
la31-vdata	6784	4611	5021	3780	6223	4959	3851	8155	6390	5418	3632	7554	5343	4553	1	1	1	1	1	1
la36-vdata	5640	3294	2677	2332	4295	2738	2230	7251	4725	2827	3205	6126	3045	2801	4	6	6	5	5	4
Number of nonconformity instances									2	2	3	3	3	4	6	6	5	5	5	4

Table 3-26: Nonconformity with SG-MFP and SG-BN as initial schedule generators.

		Makespans with SG-MCO							Makespans with SG-BN						Conformity					
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)	(17)	(18)	(19)	(20)	(21)
Instance	SG-MCO	A1	A2	A3	A4	A5	A6	SG-BN	A1	A2	A3	A4	A5	A6	A1	A2	A3	A4	A5	A6
la01-sdata	1737	1737	1588	1737	1677	1677	1677	2368	2083	1578	2117	1798	1703	1639	1	0	1	1	1	0
la06-sdata	2577	2280	2254	2500	2350	2254	2350	3004	2384	2326	2834	2257	2374	2426	1	1	1	0	1	1
la11-sdata	3653	3329	3315	3522	3092	3202	3518	3461	3200	3082	3229	2938	3003	3205	1	1	1	1	1	1
la16-sdata	3091	2347	2137	2410	2387	2407	2412	3779	2312	2307	2743	2017	2296	2501	0	1	1	0	0	1
la21-sdata	4720	3608	3368	3548	3162	3379	3432	4975	3600	3615	4096	3685	3665	3747	0	1	1	1	1	1
la26-sdata	5996	4413	4087	5007	4207	4430	4201	6119	4601	4351	5248	4767	4399	4713	1	1	1	1	0	1
la31-sdata	8654	6914	6596	6802	6736	6208	6593	9155	7601	6226	8157	6329	6293	6163	1	0	1	0	1	0
la36-sdata	6587	4504	4657	4710	3982	4145	4270	7714	4865	4328	5357	4367	4409	5133	1	0	1	1	1	1
Number of nonconformity instances									2	3	0	3	2	2	6	5	8	5	6	6
la01-edata	2320	1582	1545	1676	1550	1644	1653	2192	1631	1564	1797	1624	1685	1643	0	0	0	0	0	1
la06-edata	2605	2291	2283	2297	2096	2365	2297	3189	2295	2305	2789	2210	2239	2415	1	1	1	1	0	1
la11-edata	3586	2941	2957	3033	2839	3111	3161	3813	2952	3029	3131	3068	3022	2964	1	1	1	1	0	0
la16-edata	3392	2155	2232	2416	1804	2593	2366	3847	2001	1988	2408	2399	2420	2198	0	0	0	1	0	0
la21-edata	4630	3269	3411	3446	3223	3348	3351	5036	3820	3247	3753	3506	3480	3342	1	0	1	1	1	0
la26-edata	5544	4195	4299	4213	3318	4104	4320	6365	4711	4301	5063	4687	4521	4476	1	1	1	1	1	1
la31-edata	7982	6412	5791	6216	6133	5855	6285	9788	7435	5967	7575	6326	6042	6301	1	1	1	1	1	1
la36-edata	6373	4079	4176	4502	3105	3904	4044	8030	4557	4219	5994	4129	4247	4485	1	1	1	1	1	1
Number of nonconformity instances									2	3	2	1	4	3	6	5	6	7	4	5
la01-rdata	2206	1441	1438	1527	1550	1603	1482	2039	1426	1472	1681	1595	1588	1508	1	0	0	0	1	0
la06-rdata	2647	1942	2101	2242	2096	2085	2207	2812	2047	2168	2482	2049	2236	2175	1	1	1	0	1	0
la11-rdata	3296	2843	2908	3024	2839	2905	2987	3707	2734	2846	3038	2787	3068	2794	0	0	1	0	1	0
la16-rdata	3154	1551	1706	2016	1804	2047	1906	3398	1631	1728	1881	1880	2076	1868	1	1	0	1	1	0
la21-rdata	4557	2709	2657	3025	3223	2749	2927	4782	2852	2699	3028	2877	2610	2852	1	1	1	0	0	0
la26-rdata	5306	3488	3332	3527	3318	3627	3581	5806	3765	3175	3901	3134	3170	3760	1	0	1	0	0	1
la31-rdata	8213	6336	5799	5372	6133	6183	6197	9015	5694	6521	5660	6600	5453	5162	0	1	1	1	0	0
la36-rdata	5844	3007	2975	3296	3105	3541	3542	7833	3773	3497	4652	3582	3346	3584	1	1	1	1	0	1
Number of nonconformity instances									2	3	2	5	4	6	6	5	6	3	4	2
la01-vdata	2004	1258	1272	1423	1328	1445	1351	1867	1339	1353	1430	1360	1574	1431	0	0	0	0	0	0
la06-vdata	2594	1816	1898	2113	1793	2087	2055	2728	1828	2112	2062	1783	1937	1973	1	1	0	0	0	0
la11-vdata	3319	2529	2720	2646	2559	2634	2647	3261	2502	2648	2636	2684	2711	2637	1	1	1	0	0	1
la16-vdata	2568	1281	1394	1453	1382	1706	1435	2861	1387	1456	1579	1343	1592	1547	1	1	1	0	0	1
la21-vdata	4474	2122	2399	1886	2099	2443	2074	4593	2025	2149	2128	2818	2336	2075	0	0	1	1	0	1
la26-vdata	4991	2709	2693	2805	3603	3077	2678	5842	3181	3083	2497	4658	3203	2545	1	1	0	1	1	0
la31-vdata	7753	5279	5498	4208	7359	5448	4092	8155	6390	5418	3632	7554	5343	4553	1	0	0	1	0	1
la36-vdata	6017	2861	2918	2372	4580	2785	2759	7251	4725	2827	3205	6126	3045	2801	1	0	1	1	1	1
Number of nonconformity instances									2	4	4	4	6	3	6	4	4	4	2	5

Table 3-27: Nonconformity with SG-MCO and SG-BN as initial schedule generators.

	Best makespan with starting solutions by Initial solution generators						Best makespan with starting solutions by permutation schedules				Conformity					
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)	(17)	(18)	(19)
Instance	A1	A2	A3	A4	A5	A6	A1	A2	A3	A4	A5	A6	A1	A2	A3	A4	A5	A6
la01-sdata	1737	1578	1737	1631	1677	1639	1673	1555	1782	1561	1656	1717	0	0	1	0	0	1
la06-sdata	2280	2229	2369	2257	2254	2342	2345	2241	2596	2327	2222	2431	1	1	1	1	0	1
la11-sdata	3142	3065	3229	2938	2961	3144	3309	3108	3389	2888	3055	3088	1	1	1	0	1	0
la16-sdata	2303	2137	2351	2017	2296	2412	2345	2130	2775	2342	2484	2521	1	0	1	1	1	1
la21-sdata	3265	3368	3548	3162	3379	3142	3325	3172	3886	3144	3286	3457	1	0	1	0	0	1
la26-sdata	4409	4087	4434	4207	4262	4201	4983	4248	5073	4286	4286	4527	1	1	1	1	1	1
la31-sdata	6772	6226	6802	6329	6208	6163	7747	7651	7187	7240	5821	6032	1	1	1	1	0	0
la36-sdata	4504	4042	4681	3982	3975	4232	5439	3897	5362	3994	3809	4753	1	0	1	1	0	1
Number of nonconformity instances							1	4	0	3	5	2	7	4	8	5	3	6
la01-edata	1582	1545	1676	1550	1644	1603	1498	1494	1784	1644	1656	1717	0	0	1	1	1	1
la06-edata	2284	2132	2297	2096	2148	2212	2336	2211	2540	2279	2222	2431	1	1	1	1	1	1
la11-edata	2941	2957	3033	2839	3022	2964	3064	2973	3420	2890	3055	3088	1	1	1	1	1	1
la16-edata	2001	1988	2385	1804	2420	2198	2246	2127	2417	2270	2484	2521	1	1	1	1	1	1
la21-edata	3193	3247	3446	2945	3342	3342	3528	3184	3748	3421	3286	3457	1	0	1	1	0	1
la26-edata	3963	4155	4078	3318	4104	4006	4824	4013	4847	3872	4286	4527	1	0	1	1	1	1
la31-edata	6189	5791	6175	5479	5778	6039	7510	7636	7276	8489	5821	6032	1	1	1	1	1	0
la36-edata	3998	4027	4139	3105	3721	4044	5208	3916	5754	3779	3809	4753	1	0	1	1	1	1
Number of nonconformity instances							1	4	0	0	1	1	7	4	8	8	7	7
la01-rdata	1298	1341	1482	1490	1588	1434	1339	1342	1518	1414	1423	1563	1	1	1	0	0	1
la06-rdata	1942	2101	2242	2049	2085	2175	1937	1974	2146	1928	2323	2062	0	0	0	0	1	0
la11-rdata	2702	2846	3024	2787	2905	2757	2636	2647	2975	2455	2750	2630	0	0	0	0	0	0
la16-rdata	1550	1706	1881	1804	2047	1868	1693	1741	2125	1677	1951	1913	1	1	1	0	0	1
la21-rdata	2609	2613	2781	2379	2610	2852	2552	2602	2588	2508	2801	2748	0	0	1	1	1	0
la26-rdata	3488	3175	3527	3134	3170	3581	3807	3412	3542	3992	3930	3591	1	1	1	1	1	1
la31-rdata	5404	5560	5372	5654	5446	4883	9220	8200	5194	13079	5437	5132	1	0	1	1	0	1
la36-rdata	3007	2975	3114	2937	3346	3485	3964	3459	3637	6714	3810	3597	1	1	1	1	1	1
Number of nonconformity instances							3	3	4	4	4	3	5	5	4	4	4	5
la01-vdata	1258	1272	1423	1328	1445	1351	1254	1265	1437	1281	1350	1366	0	0	1	0	0	1
la06-vdata	1730	1803	1855	1783	1937	1973	1685	1866	1976	1868	1903	1890	0	1	1	0	0	0
la11-vdata	2411	2648	2477	2545	2506	2513	2430	2607	2689	2383	2724	2665	1	0	1	0	1	1
la16-vdata	1281	1383	1453	1343	1592	1435	1316	1360	1470	1434	1450	1331	1	0	1	1	0	0
la21-vdata	2025	2010	1886	2020	2336	2074	2113	2214	1985	4453	2277	1889	1	1	1	1	0	0
la26-vdata	2709	2693	2497	3561	3077	2545	4835	4551	2469	8549	3242	2595	1	1	0	1	1	1
la31-vdata	4611	5021	3632	6223	4959	3851	11776	9087	4240	13958	8472	7990	1	1	1	1	1	1
la36-vdata	2861	2677	2332	4295	2738	2230	6456	3321	2493	10252	3726	4483	1	1	1	1	1	1
Number of nonconformity instances							2	3	1	2	4	3	6	5	7	6	4	5

Table 3-28: Nonconformity with permutation schedules as starting solutions.

3.10 Concluding remarks

This chapter proposes a new generalized job shop scheduling model called Flexible Generalized Blocking Job Shop Problem (FGBJS) to address complex scheduling problems in practice. The FGBJS extends the classical job shop problem with four complexifying features: processor flexibility, sequence-dependent setup times, job transfer times, and blocking constraints. To solve this complex problem, this chapter proposes three constructive heuristics and six TS heuristics. These algorithms are developed by using three neighborhood structures, with which we can always find a feasible neighbor of a feasible solution by moving one operation (before another one) and resequencing the other operations of the moved operation's job if necessary. Three constructive heuristics proposed in this chapter are based on job insertion principles and have the same structure, but differ in the way a schedule is improved after a new job is appended. Six TS algorithms are based on combinations of three generic TS algorithms and the three neighborhood structures. The first generic TS algorithm is similar to the algorithm proposed by Nowicki and Smutnicki; the second one exploits a largely unattempted innovative idea of Glover to alternate between two different neighborhood structures; and the last one, considered as a hybrid of TS and Variable Neighborhood search, systematically explores different neighborhood structures in the TS framework. Our extensive computational experiments on 160 newly created benchmark instances showed that the TS algorithms gave significant improvement to initial schedules obtained by the constructive heuristics or job permutation. No TS algorithm was a clear dominator; each algorithm performed best for some specific flexibility levels and instance sizes.

Future researches on the FGBJS may aim at solving the following problems:

1. Finding a good lower bound procedure for the FGBJS, which helps us to develop ad hoc exact algorithms as well as to evaluate quality of the constructive and TS algorithms,
2. Finding fast approximation schemes to efficiently estimate the length of the longest path as opposed to the exact routine currently employed,
3. Extending the current FGBJS model and its solution methods to address more practical scheduling problems.

Chapter 4

Application in surgical case scheduling

4.1 Introduction

Surgery is an important activity in most hospitals and clinics since it is estimated to generate around two third of hospital revenues [Jac02]. On the other hand, it accounts for approximately 40% of hospital resource costs, including costs of personnel (surgeons, anaesthetists, nurses, etc.) and facilities (operating rooms, intensive care beds, etc.) [MVDM95]. Surgery takes place in a context of challenging trends such as heavy expenditure on health care [OEC03], increasing rates in health care costs [CS01], and rising surgery demand due to aging populations and technological advances that have broadened the scope of surgical interventions [Gab99].

In this context, hospital management is subject to ever mounting pressures to control surgical costs while ensuring quality of care for surgical patients in a timely manner. A successful cost containment strategy must integrate decision-making at all levels: strategic, tactical, and operational. At the operational level, one of the main problems is *surgical case scheduling (SCS)* [Gab99].

Although benefits of efficient scheduling systems are publicized in many industrial applications, few successes have been reported in healthcare. In fact, a recent survey on operating room management in Switzerland in 2001 shows that hospital management is not satisfied with the current SCS practice. Only 26% of the survey interviewees are somewhat happy with the scheduling systems, while 31% are not happy and 29% are strongly dissatisfied [SL02].

This chapter investigates the SCS problem and proposes an integrated solution approach to the problem. It is structured into six sections. Section 4.2 describes various SCS problems. Section 4.3 presents a literature review. Section 4.4 proposes a *mixed integer linear programming (MILP)*

model for SCS and discusses the model. This model is based on a novel extension of the well-known job shop scheduling problem. Section 4.5 presents a heuristic scheme that decomposes a weekly SCS problem into a series of daily SCS problems and then solves each daily SCS problem heuristically. This section then shows how to use the research results obtained from Chapter 3 to solve the daily SCS problem. Section 4.6 concludes the paper.

4.2 Problem descriptions

Having been briefly introduced in Chapter 1, the SCS problem is stated in this section in more detail. Surgical services are offered at both hospitals and *ambulatory surgical centers (ASC)*. Patients in hospitals are called *inpatients* and patients in ASCs are called *outpatients*. Typically, outpatient cases are shorter, less complex, and less variable than inpatient cases. Outpatients often have same-day surgery and do not stay overnight in ASCs, while inpatients are hospitalized for one or more days before surgery and stay in the hospital after surgery for continuing care. Many hospitals are *integrated hospitals* that serve both patient types by the same facilities [HZ98]. In such hospitals, the *ambulatory surgical unit (ASU)* is the front-line unit for outpatients.

Surgical cases are performed in *operating room* (OR) suites, including ORs where patients are operated and ORs' supporting facilities. During a surgery, a typical patient flow passes through three stages: the preoperative, perioperative, and postoperative stage. The patient flow in integrated hospitals (see Figure 4-1) can be described as follows:

- *Preoperative stage:* Inpatients are transported from nursing units to the *preoperative holding unit (PHU)* while outpatients come to PHU from the hospital's ASU. When a patient arrives at PHU, a nurse checks the patient's documents and prepares the patient. The patient is then moved to an OR.
- *Perioperative stage:* In the OR, the patient is anesthetized for surgery by an anaesthetist and then operated by one or several surgeon(s) with the assistance of one or several nurse(s) and surgical technologist(s).
- *Postoperative stage:* After surgery, the patient may be transported to several different destinations. Most patients are taken to the *postanesthesia care unit (PACU)* where they recover from residual effects of anesthesia under the care of PACU nurses. Critical inpatients (e.g. cardiac or thoracic patients) are moved directly to the *intensive care unit (ICU)* where they benefit from specialized equipment and specially trained nurses. After their stay in PACU, inpatients return to their nursing units while outpatients go through a second recovery stage in ASU before being discharged. Some outpatients might be admitted to the hospital if their health condition requires it. Other outpatients having minor regional anesthesia may bypass PACU.

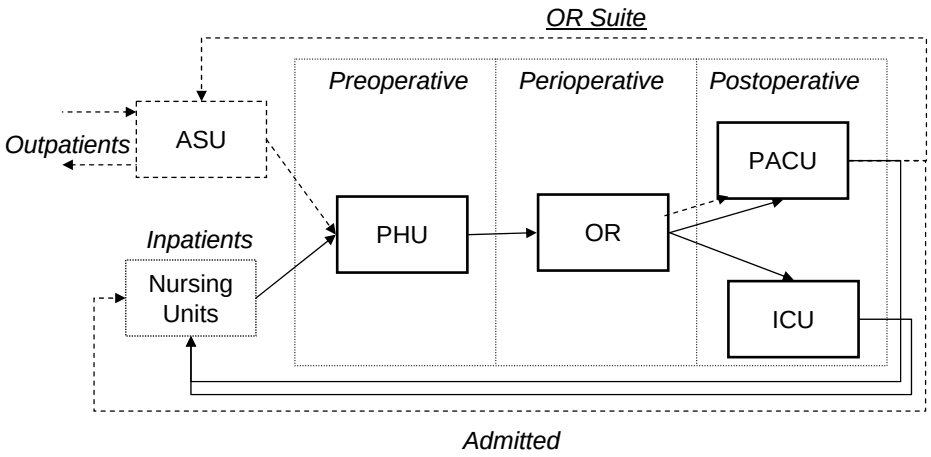

Figure 4-1: Patient flows in integrated hospitals.

Somewhat simpler than in integrated hospitals is the outpatient flow in ASCs as depicted in Figure 4-2, where the preoperative and the second-phase postoperative procedures take place in the *ambulatory unit (AU)*.

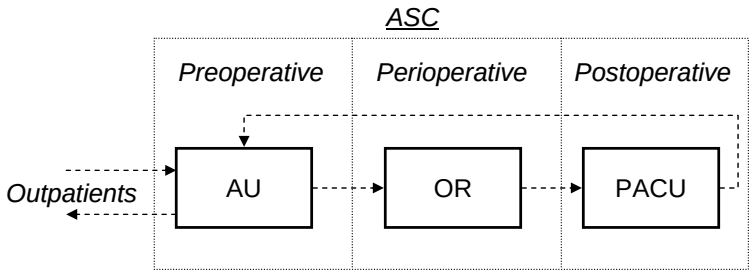

Figure 4-2: Patient flows in ambulatory surgical centers.

To control the flow of patients, SCS decides on the resource assignment and sequence of the cases in a short-term time horizon. Two questions that SCS addresses are:

- *How to allocate* hospital resources to surgical cases?
- *How to schedule* surgical cases on allocated resources?

For the first question, a set of suitable resources is assigned to each operation at any surgical stage. For the second question, a sequence of operations on each assigned resource is determined, and so are the planned starting time and ending time of each operation of the sequence.

Note that SCS, however, does not cover such planning issues as which surgical services are to be provided and which resources to be made available. Answers to these issues are inputs into SCS whose outcome can be fed back to adjust certain planning decisions [Bak74].

SCS must deal with different types of cases of different priority and predictability. In *elective cases*, inpatients or outpatients can typically wait for their surgeries for at least three days and are scheduled one or more days before the day of surgery. *Add-on cases* [Gab99] include *emergency cases* that require surgery in a very short time (less than two hours) to avoid loss of life or great harm to the patients, *urgent cases* that need attention within 24 hours to avert health deterioration and longer stay in hospital, and *add-elective cases* that are elective cases submitted to fill up the remaining OR time. Depending on their nature, add-on cases are scheduled accordingly, e.g. immediately for emergency cases in the day of surgery or after some *cut-off* time (e.g. 14:00 of the current day) for add-elective cases to perform the next day.

SCS systems fall into two main categories:

1. *Block systems*: Cases are scheduled in OR time blocks. An OR block is an OR time interval of typically a half or a full day. OR blocks can be allocated to individual surgeons or surgical groups or remain *open* to all services on a *first-come-first-served (FCFS)* basis. The allocation is presented in a 1 or 2-week cyclic timetable called *OR block allocation table* or *master surgical schedule*. An example of an OR block allocation table for a 3-OR hospital is given in Table 4-1 with 15 blocks allocated to three surgical groups.

	Mon	Tue	Wed	Thu	Fri
	8:00-16:00	8:00-16:00	8:00-16:00	8:00-16:00	8:00-14:00
OR_1	Orthopedic	General	Orthopedic	Orthopedic	Orthopedic
OR_2	Open	General	General	Plastic	General
OR_3	Plastic	Plastic	Plastic	Plastic	Open

Table 4-1: Example of OR block allocation table.

2. *Non-block systems:* Cases are scheduled on a FCFS basis at surgeons' requests. Non-block scheduling systems have turned out to have less utilization and more case cancellation than do block scheduling systems. Besides, surgeons often do not prefer these systems as their scheduled cases may scatter throughout the surgery day. Therefore, non-block systems are rarely used in health care entities [Gab99].

4.3 Literature review

SCS can be viewed as a part of a broader process which can be called *surgical process scheduling*. A classical reference is the paper by Malgerin and Martin [MM78] where surgical process scheduling comprises two steps. First, *advance scheduling* gives patients some future date for surgery. Second, *allocation scheduling* determines the sequence and resource assignment of the cases in a given day. While allocation scheduling is within the scope of SCS, advance scheduling is not. It is indeed a *case planning* process which ensures that capacity requirements for the limited resources are met and optimizes patient waiting times as well as resource effectiveness

over an intermediate-term time horizon. This case booking is in some sense similar to *master production scheduling* in production management [GC03]. Another planning problem not covered by SCS is OR block allocation to services and surgeons [BD02]. The solution of this *resource planning* problem constrains SCS decision making. Therefore, this review does not include advance scheduling and OR block allocation. In a conceptual framework proposed by Blake and Carter [BC97], surgical process scheduling includes advance, allocation, and external resource scheduling addressed at strategic, administrative, and operational decision making levels. SCS as defined in Section 2 covers allocation and external resource scheduling at the operational level in this framework.

SCS literature with quantitative orientation is relatively scarce. This brief review classifies the SCS literature according to outpatient, inpatient, and add-on case scheduling.

Although outpatient booking in clinics receives wide research attention [CV03], ASC surgical case scheduling does not. In one of the few studies on this subject, Lee, Matta, and Hsu [LdMH02] modeled an ASC as a two-stage no-wait flow shop. The first stage is the OR with surgeons as its main resources, and the second stage is the PACU with nurses. To minimize the number of PACU nurses and the makespan, they propose a heuristic approach which solves two subproblems interactively. The first finds the minimum number of PACU nurses subject to a given upper bound of the makespan, and the second minimizes the makespan at a fixed number of PACU nurses.

Inpatient case scheduling receives more attention than outpatient case scheduling does in the literature. In [GC03], inpatient scheduling is also modeled as a 2-stage no-wait flow shop, but no solution approach is attempted. Ozkarahan [Ozk95] proposed a MILP model to assign cases to ORs in order to minimize the sum of ORs' undertime and overtime costs, and then sequenced the loaded cases according to some priority rules. Sieber, Tobin, and Mcgurk [STM97] introduced a discrete-time mixed integer nonlinear programming model which uses a weighted penalty function taking into account patients' age, equipment usage conflict, and OR usage collision to assign OR time slots to patients. As their model is too hard to be solved optimally, they propose a simulated annealing heuristic approach. For any given case sequence, Weiss developed an associated cost function depending on waiting and idle times, in order to estimate cost optimal starting times of the procedures [Wei88]. He also used simulation to sequence cases and found that a best sequence is achieved by ordering cases according to increasing variance. He mentions the problem of assigning OR to cases but does not pursue it further. Also by simulation, Kuzdrall, Kwak and Schmidtz [KKS76] found the longest-case-first rule to yield the highest OR utilization rate among different dispatching rules. Dexter and Traub [DT02] discussed two simple heuristics to load cases into unfulfilled ORs as early and as late as possible, called earliest-start-time and latest-start-time respectively, provided that the surgeons and patients can choose their surgery days and no case is turned down in any day even if overtime is required. Simulation results in [DT02] suggest that earliest-start-time rule provides good schedules to reduce OR overtime. Dexter et al. [DEMdM05] evaluated strategies to reduce

delays in admission into a PACU from ORs. These delays, due to full or insufficiently staffed PACU beds, cause "blocking" in the ORs. A best practice, according to the study, is to adjust PACU staffing on the day of surgery by asking nurses to work overtime or getting help from qualified nurses of other departments, since the benefits of having scheduled cases performed outweigh the costs of working overtime due to PACU admission delays.

Literature on add-on case scheduling is again limited. Dexter, Macario, and Rodney [DMR99] used bin-packing heuristics to schedule add-elective cases into ORs in order to maximize OR utilization. On-line heuristics consider each add-on case in the order in which cases are submitted, while off-line heuristics pool add-elective cases until a cut-off time and prioritize cases by their surgical duration. Two basic on-line heuristics are *best-fit* (each case is scheduled to an OR having the least remaining time available) and *worst-fit* (each case is scheduled to an OR having the most remaining time available). Two basic off-line heuristics are *best-fit-descending* and *worst-fit-descending*, resulting from the application of on-line best-fit and worst-fit on a list of cases sorted according to their descending durations. An additional condition called "fuzzy" accepts OR overtime of 15 min to create more extended heuristics. Simulation experiments showed that the best-fit-descending-with-fuzzy-constraints heuristic achieves the best OR utilization. To sequence add-on urgent cases, Dexter, Macario, and Rodney [DMT99] considered three policies which are based on: (1) minimization of the average length of time each surgeon and patient waits, (2) FCFS basis, and (3) medical priority. They recommended that regardless of the chosen policy, the sequence should meet all medical deadlines.

Finally, we are not aware of literature on SCS in *integrated* hospitals. This review concludes with the following remarks:

1. *OR or OR suite:* Many papers focus on OR when discussing SCS. However, efficiency gains might be achieved by considering not only OR, the hospital's cost center, but also its adjoining units. Indeed, there are strong interactions between OR and preoperative and postoperative facilities as shown in the patient flow description in Section 2. This is confirmed by an efficiency study [WEE03] that reveals non-emergency causes of OR wasted time (including OR idle time and overtime) and their contribution (Table 4-2).

Units	Share of OR wasted time		Causes
Preoperative (PHU)	17%		Unprepared patients
Perioperative (OR)	65%	*including*	
		10%	Surgeon unavailability
		30%	Nurse shortage
		10%	anaesthetist shortage
		15%	Prolonged turnover time
Postoperative (PACU and ICU)	15%		Congested PACU
Transport	3%		Peak number of patients

Table 4-2: Causes of OR wasted time

The interaction can also be inferred from the ICU rejection rate, which can mount up to 24% for elective cases [KHYB00]. Without close coordination with ICU, a scheduled elective case can be rejected on its day of surgery due to a full ICU, resulting in unused OR time and negative impact on the patient. Hence, only with regard to *the whole OR suite's* activities can an efficient use of resources be achieved. These interactions have thus far not received much attention in the literature. Only a few consider together OR and PACU, while the interaction with PHU does not seem to have been addressed systematically up to now.

2. *Multi-resources:* The study of Weinbroum [WEE03] also indicates the importance of having all necessary resources during any operation. However, with some exceptions, e.g. [STM97], most studies consider the use of only one resource (ORs or nurses) during any operation and not the simultaneous use of multiple resources.

4.4 MILP model for SCS

This section models the elective surgical case scheduling problem as a *multi-mode blocking job shop (MMBJS)*, a new extension of the job shop scheduling problem, and develops a *MILP* formulation for the MMBJS.

4.4.1 Multi-mode blocking job shop problem

As shown in Chapter 2, the classical job shop scheduling problem (JS) cannot be used straightforwardly to model many industrial scheduling problems because its does not take into account a number of practical constraints. Several single-feature JS extensions and a two-feature extension (the multimode job shop MMJS) were already proposed in Chapter 2. Chapter 3 further extended the JS with a model called *flexible generalized blocking job shop* (FGBJS), which adds to the JS four practical features namely processor flexibility, sequence-dependent setup times, job transfer times, and blocking constraints. The FGBJS model was shown to be capable of modeling various complex scheduling problems in practice. However, it cannot be used to directly address the SCS problem as stated in the previous section because of the two following features of the SCS:

1. There are time windows for resources; and
2. Each operation may require several resources to process.

For this reason, we propose another novel extension of the JS problem called *multi-mode blocking job shop (MMBJS)* in order to model the SCS. The MMBJS is a result of integrating the multi-resource and time windows requirements into the FGBJS model. The MMBJS is described as

follows. There are n jobs to be performed on m resources. Each job consists of a sequence of operations. The execution of an operation requires a *set* of resources. Such a set of resources is called a *mode*. The modes of any two consecutive operations of any job have no common resources. There might be more than one mode available for each operation, justifying the term *multi-mode*. Once a mode is chosen for an operation, its processing entails a processing time during which the resources of the mode are occupied simultaneously, and possibly a setup time and a cleanup time. All these times can depend on the mode. Resources are not always available all the time. Therefore, a time interval during which the mode is available is attached to each mode. There is no buffer between resources, so that if a job has finished an operation and the job's next operation cannot be started, the resources of the finished operation remain blocked until the next operation is started. Because of this *blocking*, one needs to distinguish between the completion time of an operation, which equals the sum of its starting time and processing time, and its departure time (the time the job leaves its current processing stage to enter its next stage or leave the system). Scheduling the jobs means to assign a mode and determine the starting and leaving time for each operation in such a way that some objective function is minimized.

The next section shows how SCS can be modeled in the framework of the MMBJS, and gives a MILP formulation of the MMBJS. In subsection 4.4.3, the application of the MMBJS model to SCS with regard to several aspects in the surgical environment is discussed in more detail.

4.4.2 MILP model for SCS based on the MMBJS

A surgical case (*job*) can be considered as a sequence of processing steps (*operations*) to be performed using a certain set of hospital resources. According to the patient flow description given in Section 4.2, a case typically comprises *three* steps, corresponding to the preoperative, perioperative and postoperative stage.

The resources needed to perform a surgical case comprise personnel (surgeons, anaesthetists, nurses, ...) as well as facilities (ORs, PHU beds, PACU beds, ICU beds, specialized equipment, ...). Each processing step needs a specific set of resources for its execution. A possible choice of resource set for a processing step is in fact a *mode*. For example, a mode of a perioperative step may comprise several surgeons, one anaesthetist, one or several nurses, one OR and possibly some specialized equipment. Typically, there exist several alternative modes (*multi-mode*) for each processing step, corresponding to, for instance, the choice of different ORs for a surgery team in the perioperative step or the choice of different PACU beds in the postoperative step for a case. In practice, the number of feasible modes for any operation is quite limited for technical and organizational reasons, which will be further discussed in subsection 4.4.3.

Each mode has an associated *availability interval* specifying the time window during which all of the mode's resources are available. If the resources of a mode are *together* available in several distinct time windows, different modes consisting of the same resources for the initial mode are

introduced, one for each time window.

It is assumed that durations of the preoperative, perioperative, and postoperative steps are dependent on the chosen mode, that setup and cleanup time for each step are case-dependent, and that all these durations are deterministic and known in advance. In plus, assume that transporters are always available and transport times are negligible because of the proximity of facilities.

Preoperative and perioperative steps of a case are assumed to be *blocking operations* in the sense that the resources used for a step are not released (i.e. are blocked) until the case enters the next step and a possible cleanup is done. For example, after finishing a surgical operation in an OR, the room remains occupied until the patient can be moved to the PACU. A reserved bed in the nursing ward for any inpatient after surgery is assumed, hence postoperative operations are non-blocking.

The aim of SCS is to determine a schedule for a given set of surgical cases, that minimizes some objective function. In line with high resource utilization, the objective considered here involves the makespan, i.e. the departure of the last case. A schedule defines for the processing steps of each case the chosen mode, as well as the starting times and the leaving times.

A MILP formulation for the MMBJS can inherit the formulations of the following JS extensions presented in Chapter 3: (1) the blocking job shop (BJS), (2) the multimode job shop (MMJS), and (3) the job shop with processor time windows (JSTW). The formulation is presented as follows.

Denote by $\mathcal{J}$ the set of jobs (surgical cases). A job $J \in \mathcal{J}$ corresponds to a sequence $J = (J_1, ..., J_{|J|})$ of operations (processing steps) where J_k is the k-th operation of J. The set of all operations is given by $I = \bigcup_{J \in \mathcal{J}} J$. Operations i and j are consecutive operations of job J if $i = J_k$ and $j = J_{k+1}$ for some k, $1 \leq k < |J|$, and the set of these $|J| - 1$ ordered pairs (i, j) is denoted by O_J. J^i denotes the job to which operation i belongs, i.e. $J^i = K$ if $i \in K$, $K \in \mathcal{J}$.

M is the set of all hospital resources needed for processing the surgical cases. Any operation $i \in I$ requires a subset of resources (a *mode*), which are occupied simultaneously during the processing of i. Possible modes of operation i are given by $M_i^r \subseteq M$, $r \in R_i$, where r is the mode index and R_i is the set of mode indices associated to operation i. The set of all mode indices is $R = \bigcup_{i \in I} R_i$. For instance, there are three resources $M = \{m_1, m_2, m_3\}$ and four modes indexed by $R = \{1, 2, 3, 4\}$. Then $R_2 = \{1, 4\}$ means that operation $i = 2$ has two possible modes M_2^1 and M_2^4, where $M_2^1, M_2^4 \subseteq M$, e.g. $M_2^1 = \{m_1, m_2\}$ and $M_2^4 = \{m_1, m_3\}$. Two different modes M_i^r and M_j^s are called *incompatible* if $M_i^r \cap M_j^s \neq \emptyset$, i.e. if they have some resource in common. Each mode has an associated availability interval $[e^r, f^r]$, $0 \leq e^r < f^r$, $r \in R$, where e^r is the starting time and f^r the ending time of the mode's interval.

For any operation $i \in I$, p_i^r denotes the processing time if mode $r \in R_i$ is chosen for operation i, and p_i^{su} and p_i^{cl} respectively are the operation's setup and cleanup times. All resources of the

chosen mode r for operation i participate in the operation's setup and cleanup through their different activities.

Finally, τ denotes a dummy finish operation that goes after everything else is done and is used to measure the makespan, H a large positive number, and α a small positive weight factor.

The following decision variables are used. x_i is the starting time of operation $i \in I \cup \{\tau\}$, and l_i the leaving time of $i \in I$ (when the job's operation leaves its current processing step). For any pair of operations $i, j \in I$, $i < j$, a binary variable y_{ij} indicates whether operation i is processed before j ($y_{ij} = 1$) or operation j is processed before i ($y_{ij} = 0$), if the modes assigned to i and j are incompatible. Finally, z_i^r is a binary variable indicating whether mode $r \in R_i$ is assigned to operation i ($z_i^r = 1$) or not ($z_i^r = 0$).

Table 4-3 summarizes the notations used in the MMBJS formulation.

Sets	
M	Set of resources
I	Set of operations,
$\mathcal{J}$	Set of jobs, $\mathcal{J} \subseteq 2^I$ is the partition of I
O_J	Set of pair of consecutive operations of a job $J \in \mathcal{J}$
	$O_J = \{J_k, J_{k+1} : k = 1, \ldots, \lvert J\rvert - 1\}$
R_i	Set of indexes of modes for operation $i \in I$
R	Set of indexes of modes, $R = \bigcup_{i \in I} R_i$
M_i^r	Mode r for operation $i \in I, r \in R_i, M_i^r \subseteq M$
Parameters	
p_i^r	Processing time of operation i under mode M_i^r, $i \in I, r \in R_i$
p_i^{su}	Setup time of operation $i \in I$
p_i^{cl}	Cleanup time of operation $i \in I$
b_i	Maximum waiting time allowed for operation $i \in I$
	before the operation's job is moved to a next stage
e^r	Starting time of the availability interval of mode $r \in R$
f^r	Ending time of the availability interval of mode $r \in R$
H	Huge number
α	Very small weight factor
τ	Dummy operation of zero duration, τ is after all operations
J^i	Job to which operation $i \in I$ belongs to
Decision variables	
z_i^r	=1 if mode M_i^r is assigned to operation i, $z_i^r = 0$ otherwise,
	$i \in I, r \in R_i$
y_{ij}	=1 if operation j is processed after operation i on some
	shared resource, $y_{ij} = 0$ otherwise, $i, j \in I, i < j$
x_i	Starting time of operation $i \in I$
l_i	Leaving time of operation $i \in I$
x_τ	Starting time of dummy operation τ

Table 4-3: Notations for the MMBJS's MILP formulation.

MMBJS's MILP formulation:

$$\text{Minimize } x_\tau + \alpha \sum_{i \in I} x_i, \text{ subject to:} \tag{4-1}$$

$$\sum_{r \in R_i} z_i^r = 1 \quad \text{for all } i \in I \tag{4-2}$$

$$l_i - x_i - p_i^r z_i^r \geq 0 \quad \text{for all } i \in I, r \in R_i \tag{4-3}$$

$$l_i - x_j = 0 \quad \text{for all } (i,j) \in O_J, J \in \mathcal{J} \tag{4-4}$$

$$x_j - l_i + H(2 - z_i^r - z_j^s) + H(1 - y_{ij}) \geq p_i^{cl} + p_j^{su} \tag{4-5}$$
$$\text{for all } i,j \in I, i < j, J^i \neq J^j, r \in R_i, s \in R_j, M_i^r \cap M_j^s \neq \emptyset$$

$$x_i - l_j + H(2 - z_i^r - z_j^s) + H y_{ij} \geq p_j^{cl} + p_i^{su} \tag{4-6}$$
$$\text{for all } i,j \in I, i < j, J^i \neq J^j, r \in R_i, s \in R_j, M_i^r \cap M_j^s \neq \emptyset$$

$$x_i - e^r z_i^r \geq p_i^{su} \quad \text{for all } i \in I, r \in R_i \tag{4-7}$$

$$\sum_{r \in R_i} f^r z_i^r - l_i \geq p_i^{cl} \quad \text{for all } i \in I \tag{4-8}$$

$$x_\tau - l_i \geq p_i^{cl} \quad \text{for all } i \in I \tag{4-9}$$

$$x_i, l_i \geq 0 \quad \text{for all } i \in I \cup \{\tau\} \tag{4-10}$$

$$y_{ij} \in \{0,1\} \quad \text{for all } i,j \in I, i < j \tag{4-11}$$

$$z_i^r \in \{0,1\} \quad \text{for all } i \in I, r \in R_i \tag{4-12}$$

Objective function (1) minimizes the makespan, given by x_τ, and at the same time forces the operations to be scheduled as early as possible by minimizing, as a second term with a small weight α, the sum of all starting times. Constraints (4-2) ensure that exactly one mode is assigned to each operation. Constraints (4-3) say that if mode r is assigned to operation i, the time lag between the starting and leaving times x_i and l_i is at least p_i^r where p_i^r is the mode-dependent processing time of i. Constraints (4-4) ensure for any two consecutive operations i, j of a job, that j starts immediately when i has left its processing step.

Constraints (4-5) and (4-6) make sure that two operations i and j do not overlap in time if their assigned modes are incompatible. Observe that both constraints are redundant if the (incompatible) modes M_i^r and M_j^s are not assigned to i and j, respectively, since then $z_i^r + z_j^s \leq 1$ and hence $H(2 - z_i^r - z_j^s)$ is greater than the right hand side because of a very large value of H. If incompatible modes M_i^r, M_j^s are assigned to i, j then constraints (4-5) and (4-6) are mutually redundant, depending on whether $y_{ij} = 1$ or $y_{ij} = 0$. For instance, if $y_{ij} = 1$, (4-5) says that j starts after the leaving time of i, with a minimum time lag $p_i^{cl} + p_j^{su}$ (cleaning and setup). The value $H = \sum_{i \in I} max_{r \in R_i}\{p_i^r\} + max_{r \in R_i, i \in I}\{p_i^r\} + max_{i \in I}\{p_i^{su}\} + max_{i \in I}\{p_i^{cl}\}$ is sufficient to keep constraints (4-5) and (4-6) redundant when necessary.

Constraints (4-7) and (4-8) ensure that the setup, processing, and cleanup of any operation are done within the availability interval $[e^r, f^r]$ of its assigned mode. Constraints (4-9) say that

the makespan is larger than any operation's leaving time plus its cleanup time. Constraints (4-10) are nonnegativity constraints while constraints (4-11) and (4-12) are binary constraints for decision variables.

4.4.3 Discussion

This section discusses the use of the MMBJS model in the context of SCS.

1. Durations:

   Preoperative and postoperative operations have case-dependent duration. Cleanup and setup times depend on the type of surgery, hence they are also case-dependent. Perioperative operations might have mode dependent durations since different surgeons can operate with different durations, depending on their skills and experience.

   All durations in the model are assumed to be deterministic and known in advance. Nevertheless, it is well known in surgical practice that durations are stochastic, and their variability poses many problems to hospital management. Still, an effective and efficient deterministic solution approach will make a contributive step in the quest for solutions to the SCS problem.

2. Modes:

   The resources constituting a mode can differ from one processing stage to another. A preoperative mode may involve a nurse as its main resource. A perioperative mode can consist of one surgeon, one nurse, one surgical technologist, one anaesthetist, one OR, and one specialized equipment. A postoperative mode can comprise a staffed PHU bed.

   The number of perioperative modes is manageable for several reasons. A patient is often operated by his or her surgeon. A surgery assisting team including an anaesthetist and nurses is normally attached to an OR on a permanent basis. Each surgeon is often allocated to some predetermined ORs and has his preferred assisting team(s). Some surgery types can only be done in their dedicated OR with specialized equipment.

   The number of preoperative modes for a case can be limited as preoperative nurses are often separated from other nurses to increase the ORs' efficiency, and their number is limited. The number of postoperative modes involving ICU beds is capped due to heavy IUC bed investment. The number of postoperative modes involving staffed PACU beds often approximates the number of ORs to a PACU bed:OR ratio which is typically from 0.75 to 1.5 [SL02]. The number of PACU bed choices for a case can be further reduced since each service group is usually allotted some particular PACU beds.

   The single availability interval of a mode is defined above as the intersection of single availability intervals of all resources in the mode. In Figure 4-3, a perioperative set of resources consists of a surgeon S and an OR; each has two availability intervals. Two

different modes are then created, each comprising the same surgeon and OR, but with its own availability interval.

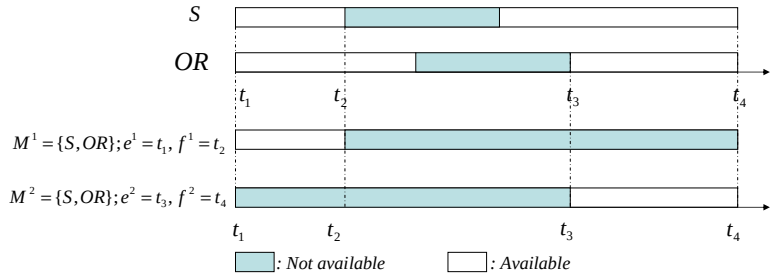

Figure 4-3: Single availability interval modes

The model assumes simultaneous use and release of all resources in any mode whereas in practice, the resources might have different starting and releasing times (e.g. a surgeon starts incision after anesthesia has been made to the patient).

3. Scheduling horizon:

   Surgical cases can be scheduled on a weekly or daily basis. The 1-day surgical schedule, which is common in the literature, is sufficient to schedule outpatients on any booked day of surgery. In addition, some services, e.g. neurology, can only schedule their inpatients one or two days in advance. But the 1-day horizon falls short for scheduling critical inpatients who need more than one day of ICU stay after their surgeries [KHYB00]. This makes the 1-day surgical schedule more vulnerable to disruptions caused by rejection due to full ICUs and less flexible for scheduling elective inpatients. Weekly schedule based on the weekly OR block allocation table is long enough to handle the ICU issue and give more choices of mode to inpatients. On the other hand, the 1-week schedule has much larger size than the 1-day schedule does, making it more difficult to solve. For both weekly and daily planning horizons, the schedule cannot remain static but needs to be updated frequently to reflect the "shop floor" status.

4. Clinical considerations:

   Defining the set of possible modes for the various operations of a case should be done with care giving absolute priority to the patient's safety and taking into account the surgeon's and assisting team's preferences. Blocking times should not be long because: (1) blocking causes poor effective resource utilization; (2) resources in an operative stage are typically not designed for activities in its downstream stage, e.g. surgical beds are not designed for recovery purposes; and (3) blocking creates crisis atmosphere in the surgical suite. Blocking time thus can be limited by the constraints:

$$l_i - x_i - \sum_{r \in R_i} p_i^r z_i^r \leq b_i \quad i \in I \tag{4-13}$$

where b_i is the maximum waiting time for i. Some hospitals require that there be *no wait* between any two consecutive operations of any case. This requirement can be met by setting $b_i = 0, i \in I$. Note that even though a patient is still recovering from anesthesia effects during the time the patient blocks an OR due to unavailability of PACU room, we should not subtract this blocking time from the patient's estimated postoperative time. The reason is that if the blocking time is counted towards the recovery time, it is implicitly accepted as productive and hence, blocking would be tolerated, which is bad for OR management as mentioned above. To further reduce the effect of blocking, we can add a penalty term $\beta \sum_{i \in I}(l_i - x_i - \sum_{k \in M_i} p_{ik})$, β is a small positive value, to the objective function (4-1). A certain urgency deadline t_i^{ub} for an operation i requires an upper bound constraint $x_i \leq t_i^{ub}$. Similarly, availability of i after some time t_i^{lb} needs a lower bound constraint $x_i \geq t_i^{lb}$.

5. Surgeons may give priority to their cases. Suppose there are two different operations $i < j \in I$, where i belongs to a case having higher priority than the case of operation j. This is modeled by setting the sequence variable $y_{ij} = 1$, which means that i precedes j on some shared resources, if any.

6. Adaptability:

   The model is suitable for both hospitals and ambulatory surgical centers (ASC) using the block scheduling systems. It can also be adapted to non-block systems by scheduling elective cases as add-elective cases in online approach (more details follow in Discussion 8).

   The model assumes fixed availability of modes and resources. This is appropriate when overtime is not allowed to perform elective surgeries, e.g. in some public-run hospitals [Man00]. However, in certain health care entities, overtime is practiced to some extent to finish the submitted elective cases. Although not directly formulated in the MMBJS model, overtime can be allowed at a given limit o^r (e.g. 4 hours) for each mode $r \in R$ in constraints (4-8):

$$\sum_{r \in R_i} (f^r + o^r) z_i^r - l_i \geq p_i^{cl}; \quad i \in I \tag{4-9'}$$

   The objective function (4-1) may add a penalty term for all cases scheduled in overtime. This penalty is calculated as $\sum_{i \in I} \gamma_i w_i$ where $\gamma_i > 0$ is the penalty factor for $i \in I$, and variable w_i represents the overtime incurred by operation i in its assigned mode r with the regular interval $[e^r, f^r]$ and satisfies the following constraints:

$$w_i \geq l_i + p_i^{cl} - \sum_{r \in R_i} f^r z_i^r; \quad i \in I \tag{4-14}$$

$$w_i \geq 0; \quad i \in I \tag{4-15}$$

An alternative is to penalize the maximum incurred overtime, denoted by a new variable w that replaces w_i in constraints (4-14) and (4-15).

In private ASCs, all planned cases should be performed on the day of surgery even in late overtime [Man00].

7. Feasibility:

   The MILP model for SCS is highly constrained in the sense that it might be impossible to obtain a feasible solution to schedule all listed cases in a planning horizon (daily or weekly) due to limited resource capacity and availability intervals of modes. To address the infeasibility issue, case planning should balance the total resource requirements and the total available resources in the first place. Even so, overtime might still be required due to the coupling of resources. When overtime is limited or not allowed, an OR manager should decide which cases to perform during the regular time. This could be done by using a *single dummy mode* as a feasibility buffer. This mode does not use any resource, is assignable to all operations, and is available from the end of the planning cycle in use. The cases having operation(s) assigned to the dummy mode are late cases and would be postponed.

8. Modifications of the MMBJS model for add-on surgical case scheduling:

   While add-on case scheduling and elective case scheduling are often separated in the literature, the proposed MMBJS model can be used to schedule add-on cases with minor modifications.

   - *Emergency case scheduling (ECS)*

     Any emergency case should be scheduled for a prompt surgery within two hours after its arrival. An add-on emergency case can delay or even bump some elective cases if their modes are incompatible with the modes assigned to the emergency. Nevertheless, the modes and sequences of scheduled elective operations should be preserved, and all in-process cases should be finished. Any bumped case due to the emergency should be performed the following day. To keep the system from being "nervous" with many changes on the following days caused by an emergency on the current day, only a "today" part of the established schedule is rescheduled. Exclusive reservation of ICU beds for emergency [KHYB00] also helps reduce the system nervousness.

     In fact, ECS can be modeled as the *job insertion problem* in the MMBJS. Given a feasible schedule of some jobs and a new unscheduled job, the job insertion problem inserts the new job into the established schedule so that the resulting schedule is feasible and some objective function is minimized [GK07]. Let I^S be the set of elective operations that are scheduled but not yet performed at the time of rescheduling, I^E the set of waiting emergency operations, $I = I^E \cup I^S$ the set of operations, $Z_i^r \in \{0, 1\}$

the mode indicator of the operation $i \in I^S, r \in R_i$, $Y_{ij} \in \{0,1\}$ the established sequence for any two incompatible operations $i, j \in I^S, i < j$, and t_i^{ub} the urgency deadline of operation $i \in I^E$. Update $e^r, r \in R$ with the scheduled departure time of the in-process operations and replace constraint (4-8) by (4-9') to allow overtime. Add to the model the following constraints:

$$z_{ir} = Z_i^r; \quad i \in I^S, r \in R_i \tag{4-16}$$

$$y_{ij} = Y_{ij}; \quad i, j \in I^S, i < j \tag{4-17}$$

$$x_i \leq t_i^{ub}; \quad i \in I^E \tag{4-18}$$

Constraints (4-16) and (4-17) preserve the mode assignment and the sequence of existing operations, respectively. Constraints (4-18) ensure the safety for emergency patients.

An OR manager could consider bumping rescheduled elective cases that would end too late and put them in the urgency list for the next day. The manager could, otherwise, try to reschedule these late cases as add-elective cases on the same day if other assignable modes are still available.

Urgent cases can be handled similarly as emergency cases with longer safety waiting times.

- *Add-elective case scheduling (ACS)*

  Add-elective cases are elective cases submitted daily to fill up OR blocks of the next surgery day, thus they can be performed or delayed. The OR manager schedules each case upon its submission in the *online scheduling* approach, or pools and schedules all submitted cases after a certain cut-off time in the *offline scheduling* approach. The new case(s) should be *inserted* into the established schedule in such a way that the chosen mode, starting and leaving time of the scheduled operations do not change. Both online and offline ACS can be formulated by setting $I = I^S \cup I^A$ (I^A is the set of add-elective operations, I^S is the set of scheduled operations) and adding to the MMBJS model constraints (4-16), (4-17), and the following constraints:

$$x_i = X_i; \quad i \in I^S \tag{4-19}$$

$$l_i = L_i; \quad i \in I^S \tag{4-20}$$

  where X_i and L_i are the respective starting and leaving times of the scheduled operation $i \in I^S$.

9. The perioperative times used in the model are in accordance with the procedural times glossary proposed by the American Association of Clinical Directors (AACD) [AAC07] as follows (abbreviations in parentheses are by the AACD). Starting time x_i corresponds

to Patient-In-Room (PIR) and leaving time l_i to Patient-Out-of-Room (POR), while processing time p_i^r is the duration from Patient-In-Room time to Procedure-Finish (PF) time. The setup time p_i^{su} is Room-Setup-Time (RST) measured from Room-Setup-Start (RSS) time to Room-Ready (RR) time; cleanup time p_i^{cl} is Room-Cleanup-Time (RCT) measured from Patient-Out-of-Room time to Room-Clean-Finished (RCF) time. If no time lag is planned between two consecutive operations and patients are assumed to be brought in the OR at Room-Ready time, then the sum $(p_i^{cl} + p_j^{su})$ for two consecutive operations i and j equals to their turnover time, which is defined as time from the prior operation's Patient-Out-of-Room time to succeeding operation's Patient-In-Room time. For each perioperative mode r, e^r and f^r correspond to Room-Open and Room-Close time respectively.

4.4.4 Computational experiments with the SCS' MILP formulation

Example 4.1.

Consider a hypothetical example of a small integrated hospital with two ORs $\{OR_1, OR_2\}$, one staffed PACU bed $\{P\}$, and one staffed ICU bed $\{IC\}$. The ORs are open from 8:00 to 16:00. The PACU bed is open from 8:00 to 17:00 while the ICU beds are available all the time. The hospital has three surgeons $\{S_1, S_2, S_3\}$, two anaesthetists $\{A_1, A_2\}$, two perioperative nurses $\{N_1, N_2\}$, and one preoperative nurse $\{N_3\}$. Each OR_i is staffed with one nurse and one anaesthetist $\{N_i, A_i\}, i = 1, 2$. The OR block allocation table over the next two days has been set up as shown in Table 4-4.

	Day 1	Day 2
OR_1	S_1	S_2
OR_2	S_3	Open

Table 4-4: OR block allocation table in Example 4.1.

Based on this input, 13 initial modes (No 1 - No 13) and one dummy mode (No 14) are constructed in Table 4-5.

Table 4-6 shows 10 cases planned to be performed over the two days, with information on the cases' number, assigned surgeon (if any), booked date (if any), expected operational durations (in minutes), and assignable modes. The cleanup and setup times for the perioperative operations are 10 minutes and 20 minutes, respectively. Blocking time is limited at 15 minutes.
The example's SCS model was coded in the mathematical modeling language LPL [Hür07] and solved by CPLEX solver (version 9.0). It has 511 binary variables, 61 continuous variables, and 817 constraints. Table 4-7 presents the optimal solution obtained after 10 seconds of computing time on a PC with a processor Pentium 4 2.8 GHz and 512 MB RAM. Figure 4-4 shows the corresponding Gantt chart. Observe, for instance, that blocking occurs when case 9 cannot be

Mode No	Mode's resources	Availability
Preoperative modes		
1	$\{N_3\}$	t_1 - t_2
2	$\{N_3\}$	t_3 - t_4
Perioperative modes		
3	$\{OR_1, S_1, N_1, A_1\}$	t_1 - t_2
4	$\{OR_2, S_1, N_2, A_2\}$	t_3 - t_4
5	$\{OR_1, S_2, N_1, A_1\}$	t_3 - t_4
6	$\{OR_2, S_2, N_2, A_2\}$	t_3 - t_4
7	$\{OR_2, S_3, N_2, A_2\}$	t_1 - t_2
8	$\{OR_2, S_3, N_2, A_2\}$	t_3 - t_4
Postoperative modes		
9	$\{P\}$	t_1 - t'_2
10	$\{P\}$	t_3 - t'_4
11	$\{IC\}$	t_1 - t_5
Dummy mode		
12	$\emptyset$	t'_4 - t_5

$t_0 = 0$ (start), $t_1 = 480$ (8:00, day 1), $t_2 = 960$ (16:00, day 1),
$t'_2 = 1020$ (17:00, day 1), $t_3 = 1920$, (8:00, day 2),
$t_4 = 2400$, (16:00, day 2),$t'_4 = 2460$ (17:00, day 2),
$t_5 = 7200$ (sufficiently large to cover ICU stays up to a week).

Table 4-5: Modes and their availability interval of Example 4.1.

Case	S	B	PHU		OR		PACU		ICU	
			D	M	D	M	D	M	D	M
Case 1	S_3	1	30	$\{1, 12\}$	180	$\{7, 12\}$	60	$\{9, 12\}$		
Case 2	S_3	2	30	$\{2, 12\}$	135	$\{8, 12\}$	75	$\{10, 12\}$		
Case 3	S_3	1	30	$\{1, 12\}$	180	{7,12}	60	$\{9, 12\}$		
Case 4	S_3		30	$\{1, 2, 12\}$	120	{7,8,12}	30	$\{9, 10, 12\}$		
Case 5	S_1		30	$\{1, 2, 12\}$	180	{3,4,12}			1500	$\{11, 12\}$
Case 6	S_1	1	30	$\{1, 12\}$	90	{5,12}	30	$\{9, 12\}$		
Case 7	S_1	2	30	$\{2, 12\}$	150	{4,12}	60	{10,12}		
Case 8	S_2		30	$\{2, 12\}$	135	{5,6,12}	30	$\{9, 10, 12\}$		
Case 9	S_2		30	$\{2, 12\}$	105	{5,6,12}	90	$\{9, 10, 12\}$		
Case 10	S_2		30	$\{2, 12\}$	90	{5,6,12}	90	$\{9, 10, 12\}$		

S: Assigned surgeon, B: Booked day of surgery, D: duration (in minutes), M: mode.

Table 4-6: Surgical cases with modes and processing times of Example 4.1.

moved to PACU until case 7 is discharged from there. Operations of case 4 are assigned to the dummy mode, i.e. case 4 cannot be finished during regular working hours of the two days.

Given the surgical schedule as shown in Table 4-7, suppose an emergency arrives early in day 1. The case's preparation, surgery, and PACU durations are 30, 180, and 90 minutes, respectively; it can be processed by any available surgeon in any available OR. The emergency case will be inserted into the set of scheduled cases in day 1={1, 3, 5, 6}. The modes' ending availability times in the emergency model are extended until 24:00 of day 1 to take into account all possible delayed cases due to the emergency. The resulting schedule in Figure 4-5 shows that the emergency delays other cases, and cases 3 and 5 are to be performed beyond the regular working time.

Operation	Case 1	Case 2	Case 3	Case 4	Case 5
Preoperative	$(1, 480, 510)$†	$(2, 2220, 2250)$	$(1, 675, 720)$	$(12, 2460, 2490)$‡	(1,615,660)
Perioperative	$(7, 510, 690)$	$(8, 2250, 2385)$	$(7, 720, 900)$	$(12, 2490, 2610)$‡	(3,660,810)
Postoperative	$(9, 690, 750)$	$(10, 2385, 2460)$	$(9, 900, 960)$	$(12, 2610, 2640)$‡	(11,810,2310)
Operation	Case 6	Case 7	Case 8	Case 9	Case 10
Preoperative	$(1, 510, 540)$	$(2, 1950, 1980)$	$(2, 2175, 2220)$	$(2, 2025, 2070)$	(2,1920,1950)
Perioperative	$(3, 540, 630)$	$(4, 1980, 2130)$	$(5, 2220, 2340)$	$(5, 2070, 2190)$	(5,1950,2040)
Postoperative	$(9, 630, 660)$	$(10, 2130, 2190)$	$(10, 2340, 2370)$	$(10, 2190, 2280)$	(10,2040,2130)

† 3-tuple indicating the chosen mode, starting time, and leaving time of the operation
‡ Postponed operations after the 2-day period

Table 4-7: Optimal solution with operations' modes, starting time, and leaving time of Example 4.1.

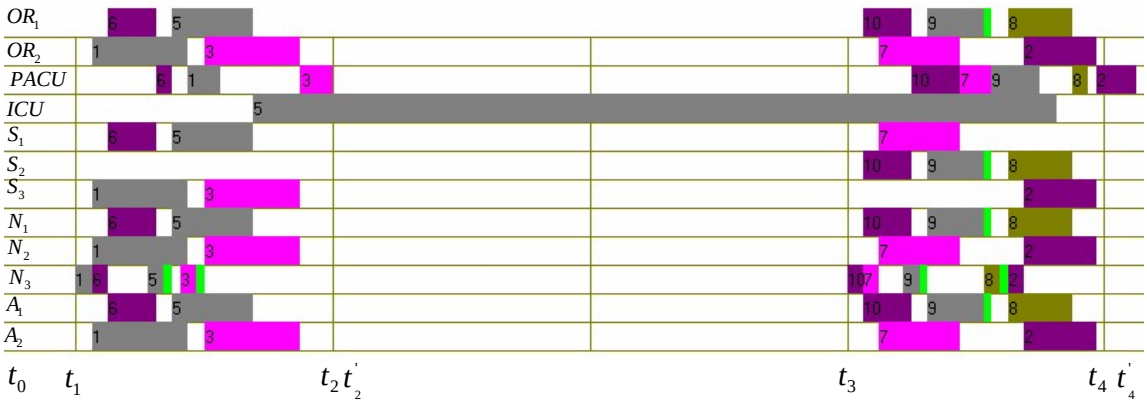

Figure 4-4: Gantt chart of Example 4.1.

Further computational experiments

Further computational experiments with practical-sized data sets include:

- Four ASC test instances labeled *a01-a04*: Four daily data sets of an ASC of six staffed ORs were obtained from the authors of [LdMH02]. Each set contains information on preassigned OR, and perioperative and postoperative durations for all cases. To complete the instances, each case assumes a cleanup time of five minutes, a setup time of ten minutes, and a maximum blocking time of five minutes. Further, six staffed PACU beds are assignable to all cases, and three PHU nurses are available for the preoperative operations. The ASC is open for eight hours per day.

- Five hospital test instances labeled *s01-s05*: The instances use data reported in [Ozk00] for real staffed OR allocation and perioperative durations (including setup and cleanup times) of selected cases during five 8-hour working days. Added information includes seven identical PACU beds, four preoperative nurses, maximum blocking time of 10 minutes, and postoperative times uniformly generated according to surgical types and perioperative durations.

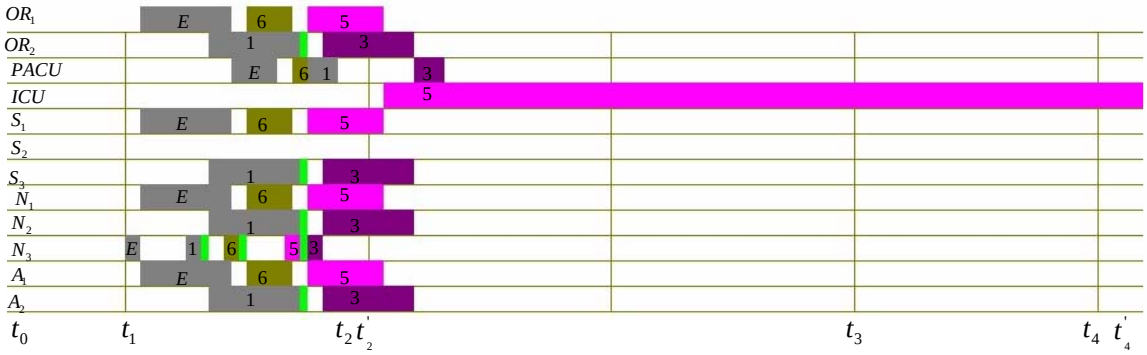

Figure 4-5: Gantt chart after an emergency arrival in day 1 in Example 4.1.

CPLEX computing times for the instances were limited at 60 minutes (see [DRLP03] for similar job shop experiments). The instances were run on a PC (Pentium 4 2.8 GHz and 512 MB RAM) with default CPLEX (version 9.0) parameters. Table 4-8 shows for each test instance the resulting MILP gap, defined as the relative gap between the obtained makespan (if any) and the lower bound given by CPLEX after the set computing time (i.e. $\frac{Makespan-CPLEX'sLB}{Makespan}$):

Number of	*a01*	*a02*	*a03*	*a04*	*s01*	*s02*	*s03*	*s04*	*s05*
Booked cases	18	25	16	18	24	25	25	26	21
Modes	16	16	16	16	21	21	21	21	21
Binary variables	1665	3100	1336	1665	2911	3151	3149	3389	2267
Continuous variables	109	151	97	109	145	151	151	157	127
Constraints	3517	6501	2829	3519	6615	7491	7471	7769	5339
Postponed cases	0	-	1	5	-	-	-	-	0
MILP gap (%)	25.5	-	0.001	0.001	-	-	-	-	0.001

-: No feasible solution was found within the allotted computing time.

Table 4-8: Results of nine SCS test instances.

Table 4-8 shows that not all planned cases could be scheduled in regular time and no optimal solution was achieved during the allotted time. This can be explained by the fact that it is already difficult to obtain feasible solutions for the BJS's MILP formulation with a general-purpose solver such as CPLEX version 9.0 (see the computational experiments in Chapter 2). Therefore, we probably need to resort to heuristic method(s) to tackle the SCS problem.

4.5 Heuristic approach to the SCS problem

4.5.1 Decomposition procedure for the weekly SCS problem

Suppose there is a list L of urgent and elective cases to be scheduled for next week of $T = 5$ working days. Assume that each elective case in L already has an admission date determined from the case planning phase and is expected to be operated no later than two days from this

date. To solve the case planning problem, one can formulate it as a mathematical programming problem (e.g. as a goal programming problem in [OE03] or an integer programming problem in [FCMA07]), then solve the resulting formulation by a general-purpose solver [OE03] or an ad hoc exact algorithm [FCMA07]. We shall not elaborate the case planning problem and its solution methods any further since it is not in the scope of this chapter. The task of the weekly SCS problem is to allocate to each planned case its necessary resources over a specified period of time within T. Preliminary computing experiments from the previous section showed that the SCS problem is very complex to be solved by an exact approach. One of successful heuristic approaches to deal with such a complex scheduling problem is to break it into more manageable subproblems (e.g. [ALM$^+$05]). We choose this approach and propose here a general procedure called *Decomp_SCS*. The Decomp_SCS decomposes the weekly SCS problem into a series of (five) daily SCS problems and then solve them sequentially; each daily SCS problem is solved heuristically. The procedure is presented below.

algorithm *(Decomp_SCS)*

begin

for each surgery day $t = 1$ to T **do**

Let U_t and E_t respectively be a set of urgent and elective cases planned for day t.

Denote by P_t a set of postponed cases in day t. Set $P_t := \emptyset$, $U_t := U_t + P_{t-1}$.

Rank the cases in E_t in the decreasing order of (1) medical priority and

(2) cost of daily care.

for each resource $r \in M$

Update its starting time of availability e^r.

Phase 1: Scheduling urgent cases

Schedule the urgent cases in U_t using some heuristic algorithm(s).

if an operation i of case $J \in U_t$ uses a resource r beyond ending availability time f^r.

then Remove all modes assignable to i containing r from R_i and **goto** Phase 1.

if $R_i = 0$ for some operation i then **stop**.

for each resource $r \in M$

Update the starting available time e^r with the scheduled urgent cases:

$e^r := l_q + p_q^{cl}$ where q is the last urgent case scheduled on r;

$e^r := e^r$ if no operation is scheduled on r.

Phase 2: Scheduling elective cases

Schedule elective cases in E_t using some heuristic algorithm(s).

19 **if** a case $J \in E_t$ uses resource r beyond its ending time f^r **then** $P_t := P_t \cup J$.

20 **end** (for)

end *(Decomp_SCS)*

Scheduling cases for each day consists of two phases. In Phase 1, after updating the starting time of availability for each resource (e.g. ICU bed No.1 is occupied until 12:00 while other resources are available from 8:00), schedule urgent cases. If an operation of an urgent case uses one of its assigned resources beyond the resource's availability interval then all modes assignable to the operation containing this resource is removed from the operation's list of assignable modes, and Phase 1 is started over. If after several iterations, a list of assignable modes for an operation is reduced to empty, then the hospital's current capacity is insufficient to handle all urgent cases. Either more capacity should be allocated by backtracking the removed modes or some urgent cases should be moved to other hospitals. After scheduling all urgent cases, update all resources' starting time of availability, then proceed with elective cases in Phase 2. Clearly, all elective cases are scheduled after the urgent cases. Finally, we check all resources' ending times of availability and postpone to the next day any elective cases that uses a resource beyond the resource's ending time of availability. These postponed cases shall be classified as urgent the next day.

In the next two subsections, the use of the FGBJS model and its solution methods in the framework of the Decomp_SCS shall be discussed.

4.5.2 Heuristic approach to the daily SCS problem in ACS

The importance of outpatient surgery industry has been steadily increasing over the past two decades. For instance, outpatient surgery in the United States accounts for more than 60% of all surgeries [Gab99]. This makes the SCS in ASC where outpatient surgeries are performed an interesting problem on its own. As introduced in Section 4.2, the SCS problem in ASCs has several particularities including:

1. Outpatient setting typically requires that all patients planned for a surgery day be operated on that day even if they are done overtime;
2. There is no ICU or night stay in ASC after a surgery;
3. Outpatient cases are often less complicated than inpatient cases are and demand less resource coordination than inpatient cases do.

Because of these ASC' particularities, when solving the SCS problem in ASC, it is reasonable to assume that:

1. Each daily SCS is independent of other daily SCS problems;

2. All resources are continuously available throughout the day from time zero;

3. There is a main resource (the so-called processor) in each of the three surgical stages (preoperative, perioperative, and postoperative) while other resources are always available upon request.

Assumption 1 allows us to investigate each daily SCS problem independently without having to apply the decomposition procedure *Decomp_SCS*. With assumption 2, the time windows constraints of the SCS can be relaxed. Further, the multiprocessor requirement is no longer necessary upon assumption 3. In fact, the processors in each stage are identified as follows: (1) preoperative nurses in the preoperative stage; (2) ORs in the perioperative stage, each OR is fully equipped and staffed; and (3) PACU beds in the postoperative stage. As a result, we can straightforwardly model the SCS scheduling problem in ASC as a FGBJS problem. Note that a sequence-dependent setup time between the two operations i and j to be performed sequentially in an OR block is typically documented in the ASC's database as a turnover time, which equals the sum of the preceding case's cleanup time and the succeeding case's setup time, i.e. $p^s_{ijk} = p^{cl}_i + p^{su}_j$ for all $k \in M_i \cap M_j$.

Example 4.2.

We applied this approach to four ASC instance *a01-a04* described in the previous section. Based the performance report of the constructive heuristics for the FGBJS in Chapter 3, we selected the constructive heuristic with the Most-Favorable-Position job insertion subroutine (the SG-MFP) as the default initial schedule generator and improved the resulting initial schedule by the $TSBJT - N^c$, a Tabu search algorithm developed in Chapter 4 that has the best overall performance on the FGBJS benchmark instances.

Obtained results are compared to corresponding results by the MILP approach (using the MM-BJS formulation with CPLEX 9.0 solver and additional overtime of three hours) in Table 4-9; all computing times were limited at 60 minutes.

ASC	MMBJS		FGBJS		Gap
case	UB	LB	SC-MFP	$TSBJT - N^c$	
(1)	(2)	(3)	(4)	(5)	(6)
a01	451.00	410.96	575.00	447.00	0.89%
a02	529.00	357.36	728.00	517.00	2.27%
a03	601.78	601.64	542.00	459.00	23.73%
a04	568.00	530.86	914.00	554.00	2.46%

Table 4-9: Results of four ASC instances modeled as the FGBJS.

Column (2) and (3) of Table 4-9 show the upper and lower bounds obtained by the MILP approach. Column (4) displays initial makespans obtained by the SG-MFP while column (5) presents improved makespans by the $TSBJT - N^c$. Values of the gaps shown in the last column, calculated as $Gap = (Upperbound(2) - Makespan(5))/Upperbound(2)$, point out that

the heuristic approach with the FGBJS model performed better the mathematical programming approach over these four ASC instances.

4.5.3 Heuristic approach to the daily SCS problem in integrated hospitals

As discussed in Section 4.2, an inpatient surgery often requires tight coordination of several resources during each of its three stages, especially in the perioperative stage where resource coordination involves a surgeon, an anaesthetist, several nurses, OR, supporting equipment, etc. Some remarks can be withdrawn from a closer look into the participating resources in each stage. First, the main capacity-constrained resource in the preoperative stage is PHU nurses since PHU beds are often available. Second, PACU beds in the postoperative stage can be considered as sufficiently nursed with different availability interval. For example, if a PACU has three supervised beds in the morning and only two supervised ones in the afternoon, we can say that the PACU has three supervised beds, two of them have a regular ending time of availability while the other has a shorter ending time of availability. If only two out of three PACU beds are staffed, then the third one has the starting time of availability equal to a huge positive value; thus this resource shall not be used by any case. The third remark is about ICU beds. An ICU bed is typically supplied with integrated supporting equipment and supervised by at least a nurse. Therefore, each recovery bed (either PACU or ICU one) can be considered as a functional processor in the postoperative stage. Finally, ORs can be seen as the main processor in the perioperative stage if the following conditions in the hospital under study are satisfied:

1. Each OR is fully staffed and installed with fixed supporting equipment;
2. Each OR's staff is capable of handling any case assigned to the OR;
3. Each patient is preassigned to a surgeon, who always accepts to operate in a medically suitable OR selected for the patient,

In this case, we can decompose the weekly SCS problem in to daily SCS problems according to the Decomp_SCS and model each daily SCS problem as a FGBJS problem. A schedule for each FGBJS problem can be first obtained by a constructive heuristic (e.g. the SG-MFP), then improved by a TS heuristic (e.g. the $TSBJT - N^c$). Note that since the FGBJS model allows the first setup time for each operation to be processor-dependent, we can handle scheduling cases with processors' positive starting times of availability as follows. Suppose an operation i is assigned to processor k having an (updated) starting time of availability e^k, then set $p^s_{\sigma ik} := e^k + p^{su}_i$. An alternative approach is to create a dummy job J^0 which has one operation on each processor k for a duration equal to the updated e^k; each operation of J^0 has to be the first one on its resource's operation sequence. After a schedule is obtained, make the following adjustments:

for each operation $i \in J^0$

if i is not the first operation on $k = \mu(i)$ **then**

Move i before the first operation on k.

end (for each)

Another technical issue is the influence of ICU stays on the makespan. An ICU stay typically lasts for more than one day, hence it dominates the makespan of a daily schedule. Therefore, the cost function computed for an associated FGBJS problem should contain not only the longest path in a problem's disjunctive blocking graph but also a sum of longest paths from the dummy start node to each operation's take-over nodes, which is equal to the sum of all operations' earliest starting times (similarly to objective function (4-1)).

Example 4.3.

(This is a modification of the SCS instance in Example 4.1.)

A hospital has one PHU nurse (N), two ORs (OR_1, OR_2), one PACU bed ($PACU$), and one ICU bed (ICU); these resources are numbered in that order from 1 to 5. Their daily available times are shown in Table 4-10.

Processor		Availability	
No.	Description	From	To
Preoperative processors			
1	N	8:00	16:00
Perioperative processors			
2	OR1	8:00	16:00
3	OR2	8:00	16:00
Postoperative processors			
4	PACU	8:00	17:00
5	ICU	Open	

Table 4-10: Processors and their availability interval of Example 4.3.

Ten cases to be operated in the next two days are listed in Table 4-11. Except urgent case 5, all other cases are elective cases at the same medical priority level.

Solve the daily SCS problem for day 1 yields the schedule depicted in the top Gantt chart in Figure 4-6, where starting time at 8:00 is counted as time zero and the full length of ICU stay for case 5 is not shown. Observe that (1) operations of urgent case 5 are the first operations on their assigned processor and (2) case 6 has its perioperative operation finished much later than the closing times of OR_1, hence this case is postponed until day 2. Continue day 2 with case 6 treated as an urgent case. In the obtained schedule (see the bottom graph in Figure 4-6), all but case 2 can be finished within the processors' time of availability. The postoperative step of case 2 exceeds the availability by only 10 minutes, thus the case is likely to be still accepted.

Case	B	PHU		OR		PACU		ICU	
		D	P	D	P	D	P	D	P
Case 1	1	30	1	180	2, 3	60	4		
Case 2	2	30	1	135	2, 3	75	4		
Case 3	1	30	1	180	2, 3	60	4		
Case 4	1	30	1	120	2, 3	30	4		
Case 5	1	30	1	180	2, 3			1500	5
Case 6	1	30	1	90	2, 3	30	4		
Case 7	2	30	1	150	2, 3	60	4		
Case 8	2	30	1	135	2, 3	30	4		
Case 9	2	30	1	105	2, 3	90	4		
Case 10	2	30	1	90	2, 3	90	4		

B: Booked day of surgery, D: duration (in min), P: processor.

Table 4-11: Surgical cases with processors and processing times in Example 4.3.

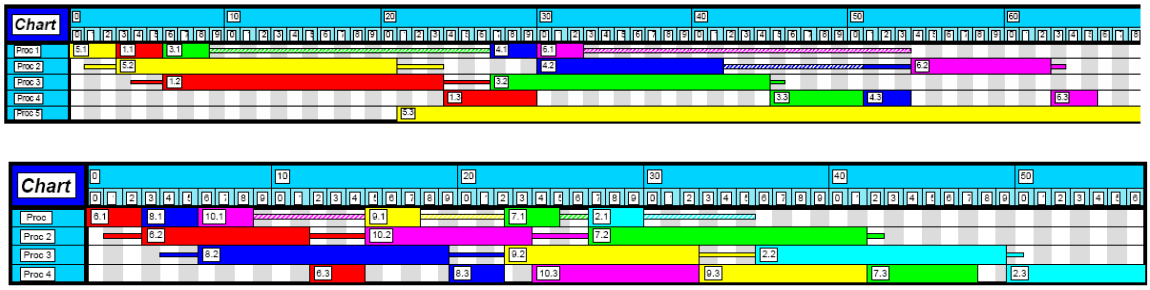

Figure 4-6: Gantt charts in Example 4.3.

4.6 Concluding remarks

This chapter identifies and analyzes the surgical case scheduling (SCS) problem including scheduling elective and add-on cases for both inpatients and outpatients. The SCS problem is modeled as a new extension of the JS called multimode blocking job shop (MMBJS). A corresponding MILP formulation is developed and preliminary computational experiments were conducted. The proposed MILP formulation for MMBJS is flexible and adaptable for SCS in many different health care settings, including both privately- or publicly-run integrated hospitals or ambulatory surgery centers, over daily or weekly scheduling periods. The main limitation in applying the MMBJS model to SCS is the capability of general-purpose MILP solvers, which for the moment can obtain feasible solutions for only small to medium-sized instances. For this reason, this paper proposes a general decomposition approach, which decomposes the SCS problem into a series of daily SCS problem if the problem's time horizon is longer than a day (e.g. a week), then tackles each single-day SCS problem heuristically. Due to several particularities of outpatient setting in ambulatory surgical centers (ASC), a daily SCS problem can be modeled as a FGBJS problem and solved independently of other daily problems in the week. On the other hand, daily SCS problems decomposed from a weekly SCS problem in an integrated hospital need to be solved sequentially, where a case using resources overtime in the day would be postponed to the next day. Under certain conditions, each daily SCS problem in an integrated

hospital can also be modeled as a FGBJS problem and solved by the methods developed in Chapter 4.

The chapter underlines the importance of connecting surgical stages in scheduling any surgical case and coordinating multiple resources during any surgical step. SCS should take a *holistic* view of all activities and resource constraints in the OR suite instead of focusing on only an individual stage like OR or ICU. Future research can be conducted in the following directions:

1. Validate the FGBJS-based solution methods with comprehensive real-life data and integrate them into a decision support system for SCS.
2. Extend solution approaches for the FGBJS to address the MMBJS problem and apply them to the SCS problem.

Chapter 5

Conclusions

In this dissertation, we studied complex job shop scheduling problems, which can take place in many industrial sectors. Four scheduling problems in manufacturing, logistics, transportation, and healthcare service have been introduced as examples of such complex job shop problems. Chapter 2 systematically studied ten practical aspects of job shop scheduling that are not covered in the classical job shop (JS) model and groups them in three categories involving jobs, processors, and job processing. This collection of features, though still incomplete, enabled us to identify different complexifying features of a given practical job shop problem and model it accordingly. Similar to any other optimization problem, a job shop extension problem can be addressed by formulating it as a mixed integer linear programming (MILP) problem and then solving it by a commercial MILP solver. We applied this approach, first by selecting experimentally the Manne formulation for the JS as a foundation upon which various formulations for single-feature job shop extensions have been developed. These formulations can be combined to formulate more complex JS problems. Our computational experiments showed that despite today's impressive performance of commercial MILP solvers when solving some optimization problems, a lot remains to be improved before one can use the mathematical programming approach with MILP formulations and commercial solvers to practically solve job shop related problems.

Chapter 3 proposed a new extended JS model which incorporates four additional features namely processor flexibility, blocking constraints, sequence-dependent setup times, and job transfer times. To our knowledge, this is the first time that blocking constraints, a practical feature that has received very high attention in research literature recently, have been addressed together with processor flexibility in a job shop-based model, which is further complexified by the presence of the other two features. This extended JS, called the Flexible Generalized Job Shop problem (FGBJS), is capable of closely modeling many practical complex scheduling problems. The FGBJS can be seen as an extension of the Generalized Job Shop (GBJS) proposed by Gröflin and Klinkert [GK05] in the flexibility dimension. We developed a MILP formulation for the FGBJS using some results in Chapter 2. As the exact approach's performance by the MILP

and CPLEX 9.0 solver was not promising, heuristic approaches to the FGBJS were deployed. We adapted the disjunctive blocking graph introduced in [GK05] to handle flexibility of a single operation so that a FGBJS instance can be solved by solving the problem of minimizing the longest path in the instance's associated disjunctive blocking graph. Basing on this graph representation, we have derived three neighborhood structures that obtain a feasible solution when moving an operation to another position. The first one of these neighborhood structures is based on evaluating positions (possibly on another processor) for the moved operation so that feasibility is ensured without making further effort. The second structure involves moving the operation to only a position on its currently assigned processor. The last neighborhood structure allows us to always obtain a feasible neighbor when moving the operation to an arbitrary position by rescheduling other operations of the operation's job if necessary. When deriving the last two neighborhood structures, we have benefited from some research results on job insertion for a generalized job shop problem by Gröflin and Klinkert [GK07]. Job insertion is also the framework of our three proposed constructive heuristics, which arrange all jobs in a certain order and then insert them one by one into a partial schedule. A resulting schedule by a constructive heuristic can be improved by one of six Tabu search (TS) heuristic algorithms that we developed. These algorithms are obtained from combining the three developed neighborhood structures with three generic TS algorithms. The first generic TS algorithm is the well-known algorithm for the JS by Nowicki and Smutnicki [NS96]. The second generic TS algorithm is based on Glover's idea to alternate between two different neighborhoods, while the last one is our new proposal that integrates TS and Variable Neighborhood Search by exploiting the presence of three neighborhood structures. We carried out extensive computational experiments on newly created benchmark instances, given the fact that the FGBJS has never been addressed before. Our computing results have shown that one constructive heuristic called the Schedule-Generator with the Most-Favorite-Position job insertion clearly outperformed the other two constructive heuristics. On the other hand, the performance of the TS algorithms varied according to many factors. Any TS algorithm can be good for a particular FGBJS instance. All six TS algorithms improved the solution obtained by the constructive heuristics substantially. In addition, the performance of a TS algorithm does not seem to depend on the quality of initial feasible solutions after some computing elapsed time.

Research results from Chapter 2 and 3 were used to address the surgical case scheduling problem (SCS) in Chapter 4. A systemwide perspective of the surgical process guided us through the studying and modeling of the SCS problem; this view is not in line with a more common approach in the SCS literature that locally focuses on the perioperative surgical stage. As a result, the SCS has been modeled as a new JS extension called the Multimode Blocking Job Shop (MMBJS), which includes into the JS four features namely multi-resource processing, flexibility, blocking constraints, and sequence-dependent setup times. We gave a MILP formulation for the MMBJS and illustrated the model with a detailed example. As the exact method with the MMBJS' formulation and a general-purpose solver is unlikely to yield optimal or even feasible

solutions within a reasonable computing time, heuristic solution methods should be sought. We showed that because of several particularities of the ambulatory surgical center's (ASC) environment, we can model the SCS problem in an ASC as a FGBJS problem and hence make use of the algorithms developed in Chapter 3 to solve the problem. This approach can also be extended to the SCS in hospitals upon certain conditions.

There remains a lot to be done from where this dissertation ends. Additional practical features of complex job shop scheduling can be included into the current feature collection. Formulations with updated big-M values, especially accompanied by advanced updating schemes, can be further studied as suggested from computational results in Chapter 2. Implementation-wise, constructive and TS heuristics developed in Chapter 3 can be further polished before they can be used as subroutines embedded in an off-the-shelf scheduling software package. Extending the research results on the FGBJS for the MMBJS is the next research step, which can substantially benefit from the job insertion framework used in developing neighborhood structures for the FGBJS. Validating the FGBJS model and its methods in an ASC or a hospital is very important to check missing constraints and application conditions. Pragmatically, one cannot expect an instant warm reception of a quantitative model in solving the SCS problem since the problem is often muddled by politics conflicts. Surgeons, who have a final say about their cases, may be hard to be persuaded to work towards a common goal of improving the efficiency of the whole OR suite. Perhaps, our best strategy is to strive for a gradual acceptance of quantitative methods in managing surgical resources.

Bibliography

[AAC07] American Association of Clinical Directors AACD, *Procedural times glossary of AACD*, http://www.aacdhq.org/Glossary.htm (2007).

[ABZ88] J. Adams, E. Balas, and D. Zawack, *The shifting bottleneck procesdure for job shop scheduling*, Management Science **34** (1988), no. 3, 391–401.

[AC91] D. Applegate and D. Cook, *A computational study of the job shop scheduling problem*, ORSA Journal on Computing **3** (1991), no. 2, 149–156.

[ALM+05] R.K. Ahuja, J. Liu, A. Mukherjee, J.B. Orlin, D. Sharma, and L.A. Shughart, *Solving real-life locomotive scheduling problems*, Transportation Science **39** (2005), 503–517.

[ANCK06] A. Allahverdi, C.T. Ng, T.C.E. Cheng, and M.Y. Kovalzov, *A survey of scheduling problems with setup times or costs*, European Journal of Operational Research **113** (2006), 390–434.

[AS05] A. Atamturk and M.W.P. Savelsbergh, *Integer programming software systems*, Annals of Operations Research **140** (2005), 67–124.

[AS06] A. Allahverdi and H.M. Soroush, *The significance of reducing setup times/setup costs*, European Journal of Operational Research **doi:10.1016/j.ejor.2006.09.010** (2006).

[AVFT+05] R. Alvarez-Valdes, R. Fuertes, J.M. Tamarit, G. Giménez, and R. Ramos, *A heuristic to schedule flexible job shop in a glass factory*, European Journal of Operational Research **165** (2005), no. 2, 525–534.

[Bak74] K.A. Baker, *Introduction to sequencing and scheduling*, John Wiley & Sons, Inc., 1974.

[BC97] J.T. Blake and M.W. Carter, *Surgical process scheduling: A conceptual literature review*, Journal of Health Systems **5** (1997), no. 3, 17–30.

[BD02] J.T. Blake and J. Donald, *Mount Sinai hospital uses integer programming to allocate OR time*, Interfaces **32** (2002), no. 2, 63–73.

[BDG06] L. Bianco, P. Dell'Olmo, and S. Giordani, *Scheduling models for air traffic control in terminal areas*, Journal of Scheduling **9** (2006), 223–253.

[BDP96] J. Blazewicz, W. Domschke, and E. Pesch, *The job shop scheduling problem: Conventional and new solution techniques*, European Journal of Operational Research **93** (1996), 132.

[BDW99] J. Blazewicz, M. Dror, and J. Weglarz, *Mathematical programming formulations for machine scheduling: a survey*, European Journal of Operational Research **51** (1999), 283300.

[Bix04] R.E. Bixby, *The sharpest cut - the impact of Manfred Padberg and his work*, MPS-SIAM Series on Optimization, vol. 4, ch. Mixed-Integer Programming - A Progress Report, 2004.

[BJS94] P. Brucker, B. Jurisch, and B. Sievers, *A branch and bound algorithm for the job shop scheduling problem*, Discrete Applied Mathematics **49** (1994), 107–127.

[BN98] P. Brucker and J. Neyer, *Tabu-search for the multi-mode job-shop problem*, OR Spektrum **20** (1998), 21–28.

[BV98] E. Balas and Vazacopulos, *Guided local search with shifting bottlneck for job shop scheduling*, Management Science **44** (1998), no. 2, 262–275.

[CS01] Economic Research & Consulting of Credit Suisse, *Le système de santé suisse - diagnostique pour un patient*, Economic Briefing **30** (2001).

[CV03] T. Cayirli and E. Veral, *Outpatient scheduling in health care: a review of literature*, Production and Operations Management **12** (2003), no. 4, 519–549.

[DEMdM05] F. Dexter, R.H. Epstein, E. Marcon, and R. de Matta, *Strategies to reduce dealys in admission into a pacu from Operating Rooms*, Journal of PeriAnesthesia Nursing **20** (2005), no. 2, 92–102.

[DMR99] F. Dexter, A. Macario, and D. Rodney, *Which algorithm for scheduling add-on elective cases maximizes OR utilization? Use of bin packign algorithms and fuzzy constraints in OR management*, Anesthesiology **91** (1999), no. 5, 1491–1501.

[DMT99] F. Dexter, F. Macario, and R.D. Traub, *Optimal sequencing of urgent surgical cases*, Journal of Clinical Monitoring and Computing **15** (1999), 153–162.

[DPP06] A. D'Ariano, D. Pacciarelli, and M. Pranzo, *A branch and bound algorithm for scheduling trains in a railway network*, European Journal of Operational Research **doi:10.1016/j.ejor.2006.10.034** (2006).

[DPRL98] S. Dauzère-Pérès, W. Roux, and J.B. Lassere, *Multi-resource shop scheduling with resource flexibility*, European Journal of Operational Research **107** (1998), 289–305.

[DPS92] R.J. Dudek, S.S. Panwalker, and M.L. Smith, *A lesson of flowshop scheduling*, Operations Research **30** (1992), no. 5, 16–22.

[DRLP03] E. Danna, E. Rothberg, and C. Le Pape, *Integrating Mixed Integer Programming and local search :A case study on job-shop scheduling problems*, 2003.

[Dro96] M. Drozdowski, *Scheduling multiprocessor tasks - an overview*, European Journal of Operational Research **94** (1996), 215–230.

[DT02] F. Dexter and R.D. Traub, *How to schedule elective surgical cases into specific Operating Room to maximize efficiency of use of OR time*, Anesthesiology **94** (2002), 933–942.

[FCMA07] H. Fei, C. Chu, N. Meskens, and A. Artiba, *Solving surgical cases assignment problem by a branch-and-bround approach*, International Journal of Production Economics **doi:10.1016/j.ijpe.2006.08.030** (2007).

[Gab99] R.A. Gable, *Operating Room management*, Butterworth-Heinemann, 1999.

[GC03] A. Guinet and S. Chaabane, *Operating theatre planning*, International Journal of Production Economics **85** (2003), 69–81.

[GK05] H. Gröflin and A. Klinkert, *A tabu search for the generalized blocking jos shop*, Internal working paper, Department of Informatics, University of Fribourg **05-13** (2005).

[GK07] ———, *Feasible insertions in job shop scheduling, short cycles and stable sets*, European Journal of Operational Research **177** (2007), 763–785.

[GKP08] H. Gröflin, A. Klinkert, and D.N. Pham, *Feasible job insertions in the multi processor task job shop*, European Journal of Operational Research **185** (2008), 1308–1318.

[GKZ93] S.C. Graves, R. Kan, and P.H. Zipkin (eds.), *Logistics of production and inventory*, Elsvier Science, 1993.

[GL97] F. Glover and M. Laguna, *Tabu search*, Kluwer, Boston, 1997.

[GLLRK79] L. Graham, L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan, *Optimization and approximation in deterministic sequencing and scheduling: a survey*, Annals of Discrete Mathematics **4** (1979), 287–326.

[Grö77] H. Gröflin, *Optimal Mehrmaschinenbelegung mit Variantenwahl am Beispiel der chemischen Chargenfertigung*, Ph.D. thesis, ETH Zürich, 1977.

[HJT94] J. Hurink, B. Jurish, and M. Thole, *Tabu-search for the job-shop scheduling problem with multipurpose machines*, OR Spektrum **15** (1994), 205–215.

[HM01] P. Hansen and N. Mladenovic, *Variable neighborhood search: Principles and applications*, European Journal of Operational Research **130** (2001), no. 2, 449–467.

[HS96] N.G. Hall and C. Sriskandarajah, *A survey of machine scheduling problems with blocking and no-wait in process*, Operations Research **44** (1996), no. 3, 510–515.

[Hür07] T. Hürlimann, *LPL: A mathematical modeling language*, http://www.virtual-optima.com/download/docs/intro.pdf (2007).

[HW03] A. Hertz and M. Widmer, *Guidelines for the use of meta-heuristics in combinatorial optimization*, European Journal of Operational Research **151** (2003), 247–252.

[HZ98] A.P. Harris and G.W. Zitzmann, *Operating Room management*, Mosby, 1998.

[ILO] ILOG, *ILOG Cplex Parameters manual*, http://www.ilog.com/products/cplex.

[Jac02] R. Jackson, *The business of surgery*, Health Management Technology **23** (2002), no. 7, 20–22.

[JM99] A.S. Jain and S. Meeran, *Deterministic job-shop scheduling: Past, present, and future*, European Journal of Operational Research **113** (1999), 390–434.

[Joh54] S.M. Johnson, *Optimal two- and three-stage production schedules with setup times included*, Naval Research Logistics Quarterly **1** (1954), 61–68.

[JRM00] A.N. Jain, B. Rangaswamy, and S. Meeran, *New and stronger job-shop neighborhoods: a focus on the method of Nowicki and Smutnicki*, Journal of Heuristics **6** (2000), 457–480.

[KE99] K.H. Kim and P.J. Egbelu, *Scheduling in a production environment with multiple process plans per job*, International Journal of Production Research **37** (1999), no. 12, 2725–2753.

[KH03] T. Kis and A. Hertz, *Lower bound for the job insertion problem*, Discrete Applied Mathematics **128** (2003), 395–419.

[KHYB00] S.C. Kim, I. Horowitz, K.K. Young, and T.A. Buckley, *Flexible bed allocation and performance in the intensive care unit*, Journal of Operations Management **18** (2000), 427–443.

[KKS76] N.K. Kwak, P.J. Kuzdrall, and H.H. Schmitz, *The GPSS simulation of scheduling policies for surgical patients*, Management Science **22** (1976), no. 9, 982–989.

[Kli01] A. Klinkert, *Optimization in design and control of automated high-density warehouses*, Ph.D. thesis, University of Fribourg, Switzerland, 2001, Nr. 1353.

[Law84] S. Lawrence, *Supplement to resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques*, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburg, PA (1984).

[LdMH97] C.Y. Lee, R. de Matta, and V.N. Hsu, *Current trends in deterministic scheduling*, Annals of Operations Research **70** (1997), 1–41.

[LdMH02] ———, *Scheduling patients in an ambulatory surgical center*, Naval Research Logistics **50** (2002), no. 3, 218–238.

[LY92] C.L. Liao and C.T. You, *An improvement formulation for the job shop scheduling problem*, Journal of Operational Research Society **43** (1992), 1047–1054.

[Man60] A.S. Manne, *On the job shop scheduling problem*, Operations Research **8** (1960), 219–223.

[Man00] OR Manager, *Efficient scheduling of OR cases*, OR Manager **16** (2000), 27–28.

[MG00] M. Mastrolilli and L.M. Gambardella, *Effective neighborhood functions for the flexible job shop problem*, Journal of Scheduling **3** (2000), no. 1, 3–20.

[MM78] J.M. Magerlein and J.B. Martin, *Surgical demand scheduling: A review*, Health Service Research **13** (1978), 418–433.

[MMR99] R. Macchiaroli, S. Mole, and S. Riemma, *Modelling and optimization of industrial manufacturing processes subject to no-wait constraints*, International Journal of Production Research **37** (1999), no. 11, 2585–2607.

[MP02] A. Mascis and D. Pacciarelli, *Job-shop scheduling with blocking and no-wait constraints*, European Journal of Operational Research **143** (2002), 498–517.

[MSB88] K. McKay, F. Safayeni, and J. Bzacott, *Job shop scheduling theory - what is relevant?*, Interfaces **18** (1988), 84–90.

[MVDM95] A. Macario, T.S. Vitez, B. Dunn, and T. McDonald, *Where are costs in perioperative? Analysis of hospital costs and charges for inpatient surgical care*, Anesthesiology **83** (1995), no. 6, 1138–1144.

[NS96] E. Nowicki and C. Smutnicki, *A fast taboo search algorithm for the job shop problem*, Management Science **42** (1996), no. 6, 797–812.

[OE03] S.N. Ogulata and R. Erol, *A hierarchical multiple criteria mathematical programming approach for scheduling general surgery operations in large hospitals*, Journal of Medical Systems **27** (2003), no. 3, 250–270.

[OEC03] OECD, *OECD annual report*, downloadable from http://www.oecd.org.

[Ozk95] I. Ozkarahan, *Allocation of surgical procedures to Operating Rooms*, Journal of Medical Systems **19** (1995), no. 4, 333–352.

[Ozk00] ———, *Allocation of surgeries to Operating Rooms by goal programming*, Journal of Medical Systems **24** (2000), no. 6, 339–378.

[Pan97] C.H. Pan, *A study of integer proramming formulations for scheduling problems*, International Journal of Systems Science **28** (1997), no. 1, 33–41.

[PR00] V. Portougal and J.D. Robb, *Production scheduling theory: Just where is it applicable?*, Interfaces **30** (2000), no. 6, 64–76.

[PW06] Y. Pochet and L.A. Wolsey, *Production planning by mixed integer programming*, Springer, New York, 2006.

[Ree93] C.R. Reeves (ed.), *Modern heuristic techniques for combinatorial problems*, Blackwell Scientific Press, Oxford, 1993.

[RJ96] S.E. Ramaswamy and S.B. Joshi, *Deadlock-free schedules for automated manufacturing workstations*, IEEE Transactions on Robotics and Automation **12** (1996), 391–400.

[RK76] A.H.G. Rinnooy Kan, *Machine scheduling problems: Classifcation, complexity and computations*, The Hague, The Netherlands., 1976.

[RS64] B. Roy and B. Sussmann, *Le problèmes d'ordonnancement ace contraintes disjonctives*, SEMA, Paris, France **9 bis** (1964).

[Sch98] J.M.J. Schutten, *Practical job shop*, Annals of Operations Research **83** (1998), 161–177.

[SL02] T.J. Sieber and D.L. Leibundgut, *Operating Room management and strategies in Switzerland: results of a survey*, European Journal of Anesthesiology **19** (2002), 415–423.

[STM97] D. Sier, P. Tobin, and C. Mcgurk, *Scheduling surgical procedures*, Journal of the Operational Research Society **48** (1997), 884–891.

[Tai93] E. Taillard, *Benchmarks for basic scheduling problems*, European Journal of Operational Research **64** (1993), no. 2, 278–285.

[TSG04] F.T. TSeng, E. Stafford, and J.N.D. Gupta, *An empirical analysis of integer programming formulations for the permutation flowshop*, Omega **32** (2004), 285–293.

[VAL96] R.J.M. Vaessens, E.H.L. Aarts, and J.K. Lenstra, *Job shop scheduling by local search*, Journal on Computing **8** (1996), no. 2, 302–317.

[VLAL92] P.J.M. Van Laarhoven, E.H.L. Aarts, and J.K. Lenstra, *Job shop scheduling by simulated annealing*, Operations Research **40** (1992), 113–125.

[Wag59] H.M. Wagner, *An integer linear programming model for machine scheduling*, Naval Research Loistics Quarterly **6** (1959), 131–140.

[WEE03] A.A. Weinbroum, P. Ekstein, and T. Ezri, *Efficiency of the operating room suite*, The American Journal of Surgery **185** (2003), 244–250.

[Wei88] N.E. Weiss, *Model for determining estiamted start times and case orderings in hospitals ORs*, IIE Transaction **22** (1988), 143–150.

[WR90] K.P. White and R. Rogers, *Job shop scheduling-limits of the binary disjunctive formulation*, International Journal of Production Research **28** (1990), no. 12, 2187–2200.

APPENDIX

.1 Scheduling terminologies

Scheduling research started with studies in manufacturing industries in the 1950s. That is why many scheduling terminologies have their root in manufacturing context. Since scheduling problems are not limited to only manufacturing environments, this dissertation applies the following general scheduling terms:

- a *job* is a set of activities that must be processed in a certain order, e.g. a customer's order, a surgery, and so on;
- a *processor* is a resource used to process jobs, e.g. a machine, a surgeon, and so on;
- an *operation* is job processing.

The most popular representation for schedules in industry is *Gantt chart*, which displays the activities as the boxes over the horizontal time axis. Below is an example of Gantt chart for a schedule.

Example .1.

Consider scheduling a job consisting of three operations 1, 2, 3, to be processed in this order; operation 1 is performed by processor A for a duration of one time unit, operation 2 by processor B for two time units, and operation 3 by processor C for three time units. The Gantt chart shows that operation 1 starts at time 0 and finishes at time 1; operation 2 starts at time 1 and finishes at time 3; and operation 3 starts at time 3 and finishes at time 6. Thus, the starting and finishing times of the operations fulfill the processing time and precedence requirements.

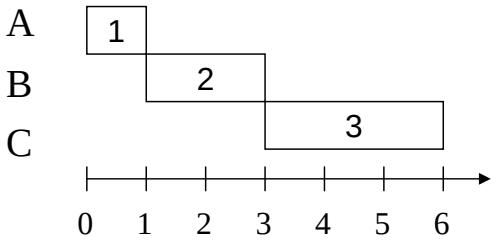

Figure -1: Example of using Gantt chart to represent a schedule.

Given a solution to a scheduling problem, a *left-shift* of an operation is possible if an operation can start earlier, i.e. a box representing the operation in the solution's Gantt chart can be shifted to the left. The left-shift is called *local* if the operation can start earlier without changing the current sequence on the processor(s) performing the operation, and *global* if otherwise. A global

left-shift of an operation may delay the starts of other operations in the same operation sequence. Suppose in Example .1 we have another job consisting of operations 4 and 5 to be processed in this order by processors B and A for two and three time units respectively. A solution is shown to the left of Figure -2. To the right of this figure is another solution where left-shifts of one time unit are applied to operations 4 and 5. These operations can start even earlier by applying a global-shift of three time units to operation 4 and a local left-shift of three time units to operation 5, respectively, as displayed to the right of Figure -3. Observe in this figure that the operation sequence of processor B is changed by the global left-shift and operations 2 and 3 are delayed.

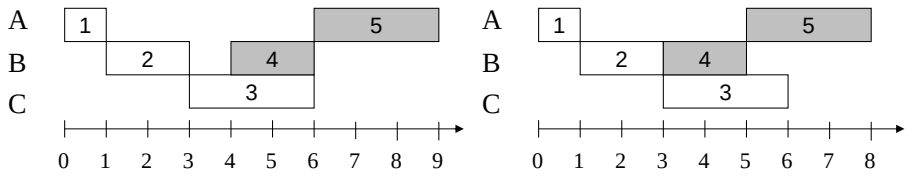

Figure -2: Example of local left-shifts.

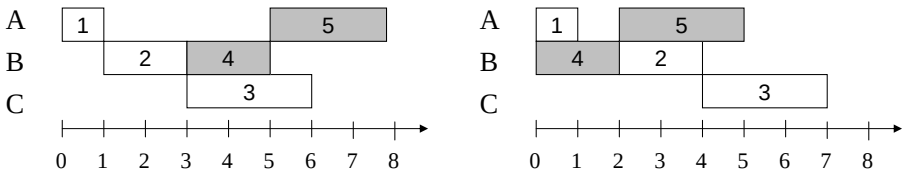

Figure -3: Example of global left-shifts.

A schedule is:

- *semi-active* if no operation can start earlier without changing the operation sequence or violating some precedence constraint,
- *active* if no operation can start earlier without delaying at least another operation or violating some precedence constraint,
- *non-delay* if no processor stays idle when there is some job ready for processing by the processor.

Thus, a semi-active schedule cannot have any local left-shift. A non-delay schedule is also an active schedule and an active schedule is also a semi-active schedule. The schedule to the left of Figure -3 is active while the one to the right is non-delay.

.2 Subroutine algorithms in Chapter 3

Let $G' = (V', E', c)$ be a disjunctive blocking digraph for a GBJS' instance, where V' is the set of nodes, E' is the set of arcs and c specifies the arc weights. Remove from E' transitive disjunctive arcs, then contract each pair of hand-over and take-over nodes of two consecutive operations of the same job to obtain a disjunctive blocking graph $G = (V, E, c)$. Define for G $n := |V|$, $m := |E|$, dummy start node $\sigma \in V$ of indegree $|\delta^-(\sigma)| = 0$ and dummy node $\tau \in V$ of outdegree $|\delta^+(\tau)| = 0$. The following two algorithms are used as subroutines in some algorithms in Chapter 3:

1. algorithm *Path_detection* to check the existence of a path from i to $j, i, j \in V$;
2. algorithm *Longest_path* to find the the length of a longest path from σ to τ or detect the existence of a cycle in the solution graph.

The details of these subroutines are given below.

algorithm *Path_detection*

Input: $G = (V, E)$ and $i, j \in V$.

Output: Yes/No answer to the question if there is any path from i to j in G.

begin

21 **if** $i = j$ **then return** *Yes.*

22 Let $R = \{i\}$. Set label $l(v) :=$ *unvisited* for all $v \in V$.

23 **while** $R \neq \emptyset$ **do**

24 Choose $u \in R$. Set $R := R - u$ and $Q := \{v \in V : (u, v) \in E\}$.

25 **for** all $v \in Q$ **do**

26 **if** $v = j$ **then return** *Yes.*

27 **else if** $l(v) =$ *unvisited* **then** set $l(v) :=$ *visited* and $R := R + v$.

28 **end** (for)

29 **end** (while)

30 **return** *No.*

end (*Path_Detection*)

Set R keeps the list of visited nodes that are reachable from i. The algorithm terminates when j is included into R or all reachable nodes from i have been checked without obtaining the positive answer. As the algorithm checks each node and each arc at most once, its complexity is $O(n + m)$. As any node in G has at most 2 entering and 2 leaving arcs corresponding to the processing and sequencing precedences, the complexity becomes $O(n)$.

algorithm *Longest_path*

Input: $G = (V, E, c)$.

Output: Detection of a cycle (if any) in G, the length of the longest path l_v from σ to each $v \in V$, and the list $P(v)$ of preceding nodes of v.

begin

Store the indegree $|\delta^-(v)|$ of each node, $v \in V$.

for all $v \in V$

Set the length of a longest path from source σ to v as $l_v := 0$

Set the predecessor list of v as $P(v) := \emptyset$.

end (for)

Set the set of nodes having zero indegree $S := \{s\}$. Set the inspected node $u := s$.

Set number of visited nodes $\gamma := 0$ and cycle detection flag $\theta := false$.

while $S \neq \emptyset$ **do**

Let u be the last element included into S.

Set $S := S - u$, $R := \{v \in V : (u, v) \in E\}$ and $\gamma := \gamma + 1$.

for all $v \in R$ **do**

if $l_v < l_u + c_{(u,v)}$ **then** set $l_v := l_u + c_{(u,v)}$ and $P(v) := P(v) + v$;

else if $l(v) = l(u) + c_{(u,v)}$ **then** set $P(v) := P(v) + v$.

Set $|\delta^-(v)| := |\delta^-(v)| - 1$.

if $|\delta^-(v)| = 0$ **then** set $S := S + v$.

end (for)

end (while)

if $\gamma < |V|$ **then** set $\theta := true$.

return θ.

end (*Longest_path*)

Zero indegree nodes in S are inspected in the first-in-first-out basis according to their order of inclusion to S.The algorithm ends when all nodes having indegree reduced to 0 have been checked. When the first node belonging to a cycle is checked, since its reduced indegree remains positive, no further node can be added to S hence the algorithm stops. Therefore, the number of nodes visited is less than $|V|$, which confirms the presence of a cycle. The computational effort is $O(m)$ since each arc is visited at most once.

.3 Performance of the FGBJS' MILP formulation

Each of the 160 benchmark instances was let run for 30 minutes with CPLEX 9.0 solver. Table -1 shows upper and bounds obtained, where mark (-) indicates that no feasible solution was obtained. The found solutions were compared with solutions obtained from the three constructive heuristics and a TS heuristic ($TSBJT - N^c$), see Tables -2 to -5. Observe that the constructive heuristics yielded better solutions for a majority of instances in much shorter computing times. The $TSBJT - N^c$, taken as a representative of the six TS heuristics, outperformed the MILP formulation with CPLEX 9.0 solver for all of the four data sets but only two *sdata* instances.

	Upper bound				Lower bound			
	sdata	edata	rdata	vdata	sdata	edata	rdata	vdata
la01	1549	1639	1625	-	1118	960.402	751	499
la02	1609	1676	1777	-	1092.001	947.381	715	581
la03	1433	1499	1501	-	1144	838	635.001	496
la04	1425	1523	1574	1347	1071	956	529.5888	493
la05	1443	1475	-	1448	914	843	526	534
la06	2596	2805	-	-	928.3659	926.16	659	549.003
la07	2518	2467	-	-	953.6024	904.796	727.001	614.001
la08	2481	-	-	-	898.8801	814	725	494
la09	2682	-	-	-	1021	916.171	660.2337	557
la10	2895	-	-	-	1047	965.001	691	601
la11	4106	5210	-	-	886	841.13	740	612
la12	-	-	-	-	776	820.891	816	525
la13	-	-	-	-	919.2961	949.43	689	596
la14	4291	-	-	-	930	854.711	624	655
la15	3671	3844	-	-	892.8378	840	748	533
la16	2444	2892	-	-	1224	1123	895	895
la17	2543	-	-	-	1091.442	1011	899	789
la18	2301	2341	-	-	1155.153	1115.55	885.002	810
la19	2885	2801	-	-	1161	1155.63	914.5387	773
la20	2472	2922	-	-	1255	1234	1038	919
la21	4821	-	-	-	1193	1131	1007	846
la22	7234	-	-	-	1089.329	1092	958	813
la23	-	-	-	-	1078	1037	881	845
la24	-	-	-	-	1118	1073.15	937	882
la25	5440	-	-	-	1149	1107.72	942	915
la26	6418	-	-	-	1129	1107	949	-
la27	10496	-	-	-	1139	1154.86	1004	-
la28	7729	-	-	-	1120	1094.66	1021	-
la29	6496	-	-	-	1102.09	1101	947.6967	-
la30	10570	-	-	-	1169	1157	1049	-
la31	14711	-	-	-	1009	1078	967	-
la32	15818	-	-	-	1067	1126.07	967	-
la33	15033	-	-	-	968.0087	968	942	-
la34	14425	-	-	-	990	997.497	933	-
la35	15038	-	-	-	1003.028	962	880.3	-
la36	11348	-	-	-	1451	1421	1289	-
la37	8835	-	-	-	1436	1417.09	1336.596	-
la38	11340	-	-	-	1401	1309.83	1220	-
la39	11651	-	-	-	1350	1325.42	1237	-
la40	11490	-	-	-	1374.024	1321.29	1275	-

Table -1: Performance of the FGBJS's MILP formulation with CPLEX 9.0.

	Makespan					Deviation = (*Heuristic* - MILP)/MILP			
	MILP	SG-MCO	SG-MFP	SG-BN	TS	SG-MCO	SG-MFP	SG-BN	TS
la01-sdata	1549.00	1737	1988	2368	1583	12.14%	28.34%	52.87%	2.19%
la02-sdata	1609.00	1949	1841	2106	1546	21.13%	14.42%	30.89%	-3.92%
la03-sdata	1433.00	1745	1542	1988	1423	21.77%	7.61%	38.73%	-0.70%
la04-sdata	1425.00	1917	1994	1952	1439	34.53%	39.93%	36.98%	0.98%
la05-sdata	1443.00	1470	1696	1878	1430	1.87%	17.53%	30.15%	-0.90%
la06-sdata	2596.00	2577	2472	3004	2229	-0.73%	-4.78%	15.72%	-14.14%
la07-sdata	2518.00	2684	2526	2751	2146	6.59%	0.32%	9.25%	-14.77%
la08-sdata	2481.00	2738	2406	2518	2239	10.36%	-3.02%	1.49%	-9.75%
la09-sdata	2682.00	2696	2717	2944	2301	0.52%	1.30%	9.77%	-14.21%
la10-sdata	2895.00	2629	2742	3428	2413	-9.19%	-5.28%	18.41%	-16.65%
la11-sdata	4106.00	3653	3526	3461	3065	-11.03%	-14.13%	-15.71%	-25.35%
la14-sdata	4291.00	3523	3446	3872	3103	-17.90%	-19.69%	-9.76%	-27.69%
la15-sdata	3671.00	3368	3399	3667	3146	-8.25%	-7.41%	-0.11%	-14.30%
la16-sdata	2444.00	3091	2886	3779	2146	26.47%	18.09%	54.62%	-12.19%
la17-sdata	2543.00	2818	2430	3451	2083	10.81%	-4.44%	35.71%	-18.09%
la18-sdata	2301.00	2964	3098	3349	2227	28.81%	34.64%	45.55%	-3.22%
la19-sdata	2885.00	2887	2685	3567	2265	0.07%	-6.93%	23.64%	-21.49%
la20-sdata	2472.00	3705	2969	3644	2252	49.88%	20.11%	47.41%	-8.90%
la21-sdata	4821.00	4720	4219	4975	3606	-2.10%	-12.49%	3.19%	-25.20%
la22-sdata	7234.00	4954	3880	4561	3263	-31.52%	-46.36%	-36.95%	-54.89%
la25-sdata	5440.00	4693	4059	5187	3446	-13.73%	-25.39%	-4.65%	-36.65%
la26-sdata	6418.00	5996	4752	6119	4317	-6.58%	-25.96%	-4.66%	-32.74%
la27-sdata	10496.00	5388	5600	6985	4351	-48.67%	-46.65%	-33.45%	-58.55%
la28-sdata	7729.00	6341	5723	6601	4545	-17.96%	-25.95%	-14.59%	-41.20%
la29-sdata	6496.00	6040	5313	6228	4135	-7.02%	-18.21%	-4.13%	-36.35%
la30-sdata	10570.00	6198	5247	6596	4456	-41.36%	-50.36%	-37.60%	-57.84%
la31-sdata	14711.00	8654	7916	9155	6716	-41.17%	-46.19%	-37.77%	-54.35%
la32-sdata	15818.00	9114	8067	9632	6614	-42.38%	-49.00%	-39.11%	-58.19%
la33-sdata	15033	8251	8417	9589	6402	-45.11%	-44.01%	-36.21%	-57.41%
la34-sdata	14425	7976	8720	8832	6520	-44.71%	-39.55%	-38.77%	-54.80%
la35-sdata	15038	8605	7974	10122	6401	-42.78%	-46.97%	-32.69%	-57.43%
la36-sdata	11348	6587	5708	7714	4042	-41.95%	-49.70%	-32.02%	-64.38%
la37-sdata	8835	7143	5905	8229	4516	-19.15%	-33.16%	-6.86%	-48.89%
la38-sdata	11340	6415	5379	8038	4267	-43.43%	-52.57%	-29.12%	-62.37%
la39-sdata	11651	5603	5901	7331	4424	-51.91%	-49.35%	-37.08%	-62.03%
la40-sdata	11490	6572	5310	7767	4218	-42.80%	-53.79%	-32.40%	-63.29%
					Average =	-11.29%	-16.64%	-0.81%	-31.38%

Table -2: MILP formulation with CPLEX 9.0 vs. heuristics for *sdata* set.

	Makespan					Deviation = (*Heuristic* - MILP)/MILP			
	MILP	SG-MCO	SG-MFP	SG-BN	TS	SG-MCO	SG-MFP	SG-BN	TS
la01-edata	1639.00	2320	1909	2294	1556	41.55%	16.47%	39.96%	-5.06%
la02-edata	1676.00	1916	1795	2060	1588	14.32%	7.10%	22.91%	-5.25%
la03-edata	1499.00	1855	1666	1721	1422	23.75%	11.14%	14.81%	-5.14%
la04-edata	1523.00	1799	1782	1760	1436	18.12%	17.01%	15.56%	-5.71%
la05-edata	1475.00	1504	1740	1796	1425	1.97%	17.97%	21.76%	-3.39%
la06-edata	2805.00	2605	2611	3189	2132	-7.13%	-6.92%	13.69%	-23.99%
la07-edata	2467.00	2447	2429	2877	2128	-0.81%	-1.54%	16.62%	-13.74%
la11-edata	5210.00	3586	3500	3813	3152	-31.17%	-32.82%	-26.81%	-39.50%
la15-edata	3844.00	3374	3274	3556	3031	-12.23%	-14.83%	-7.49%	-21.15%
la16-edata	2892.00	3392	2994	3847	2252	17.29%	3.53%	33.02%	-22.13%
la18-edata	2341.00	2886	3027	3317	2065	23.28%	29.30%	41.69%	-11.79%
la19-edata	2801.00	3306	3035	3395	1921	18.03%	8.35%	21.21%	-31.42%
la20-edata	2922.00	3551	2773	3780	2170	21.53%	-5.10%	29.36%	-25.74%
					Average =	9.88%	3.82%	18.18%	-16.46%

Table -3: MILP formulation with CPLEX 9.0 vs. heuristics for *edata* set.

	Makespan					Deviation = (*Heuristic* - MILP)/MILP			
	MILP	SG-MCO	SG-MFP	SG-BN	TS	SG-MCO	SG-MFP	SG-BN	TS
la01-rdata	1625.00	2206	1908	2071	1341	35.75%	17.42%	27.45%	-17.48%
la02-rdata	1777.00	2042	1944	1812	1460	14.91%	9.40%	1.97%	-17.84%
la03-rdata	1501.00	1670	1527	1572	1235	11.26%	1.73%	4.73%	-17.72%
la04-rdata	1574.00	1637	1653	1472	1216	4.00%	5.02%	-6.48%	-22.74%
la33-rdata	15033.00	7703	8031	8335	5671	-48.76%	-46.58%	-44.56%	-62.28%
la34-rdata	14425.00	8375	7698	8012	5770	-41.94%	-46.63%	-44.46%	-60.00%
la35-rdata	15038.00	8319	7952	8464	6014	-44.68%	-47.12%	-43.72%	-60.01%
la36-rdata	11348.00	5844	5833	7262	3286	-48.50%	-48.60%	-36.01%	-71.04%
la37-rdata	8835.00	6609	5662	7988	3505	-25.20%	-35.91%	-9.59%	-60.33%
la38-rdata	11340.00	5896	5613	7310	2999	-48.01%	-50.50%	-35.54%	-73.55%
la39-rdata	11651.00	6470	5595	7654	2945	-44.47%	-51.98%	-34.31%	-74.72%
la40-rdata	11490.00	6124	5691	7587	3471	-46.70%	-50.47%	-33.97%	-69.79%
					Average =	-23.53%	-28.69%	-21.21%	-50.63%

Table -4: MILP formulation with CPLEX 9.0 vs. heuristics for *rdata* set.

	Makespan					Deviation = (*Heuristic* - MILP)/MILP			
	MILP	SG-MCO	SG-MFP	SG-BN	TS	SG-MCO	SG-MFP	SG-BN	TS
la04-vdata	1347.00	1548	1623	1658	1110	14.92%	20.49%	23.09%	-17.59%
la05-vdata	1448.00	1423	1462	1961	1206	-1.73%	0.97%	35.43%	-16.71%
					Average =	6.60%	10.73%	29.26%	-17.15%

Table -5: MILP formulation with CPLEX 9.0 vs. heuristics for *vdata* set.