



HOKKAIDO UNIVERSITY

Title	Studies on Heuristic Approach with Priority Rules for Extended Job-shop Scheduling Problem
Author(s)	横山, 想一郎
Degree Grantor	北海道大学
Degree Name	博士(情報科学)
Dissertation Number	甲第12180号
Issue Date	2016-03-24
DOI	https://doi.org/10.14943/doctoral.k12180
Doc URL	https://hdl.handle.net/2115/61589
Type	doctoral thesis
File Information	Soichiro_Yokoyama.pdf



Studies on Heuristic Approach with Priority Rules for
Extended Job-shop Scheduling Problem

Soichiro Yokoyama

Submitted in partial fulfillment of the requirements
for Degree of Doctor of Information Science
Hokkaido University

January 2016

Contents

1	Introduction	1
1.1	Background and Research Objective	1
1.2	Job-shop Scheduling Problem (JSP)	7
1.2.1	Description of the Problem	7
1.2.2	Approximation Algorithms	10
1.3	Outline of the Thesis	15
1.4	Conclusion	15
	Bibliography	16
2	Extended Job-shop Scheduling Problems	20
2.1	Introduction	20
2.2	Tardiness Minimization Job-shop Scheduling Problem	21
2.2.1	Description of the Problem	21
2.2.2	Approximation Algorithms	24
2.3	Flexible Job-shop Scheduling Problem	28
2.3.1	Description of the Problem	28
2.3.2	Approximation Algorithms	29
2.4	Conclusion	33
	Bibliography	34
3	Priority rule-based method for Tardiness Minimization	37
3.1	Introduction	37
3.2	Proposed Approximation Algorithm	39
3.2.1	Rule-based reconstruction procedure	40
3.2.2	Linear combination of priority rules	41
3.2.3	Integration with local search procedure	42
3.3	Performance Evaluation	45

3.3.1	Performance of rule-based reconstruction procedure	46
3.3.2	Comparison with existing methods	50
3.4	Conclusion	55
	Bibliography	56
4	Priority rule-based method for Flexible Job-shop Scheduling Problem	58
4.1	Introduction	58
4.2	Proposed Approximation Algorithm	59
4.2.1	PRC	60
4.2.2	Integration with a HGA	62
4.3	Perfomance Evaluation	63
4.3.1	Experiments on benchmark problems	64
4.3.2	Experiments of long-term performance on problems with various prop- erties	69
4.4	Conclusions	75
	Bibliography	76
5	Conclusion	78
	Acknowledgment	81

List of Figures

1.1	Gantt chart of an example schedule	8
1.2	Example of an active schedule	9
1.3	Example of a semi-active schedule	9
1.4	Example of a disjunctive graph for the JSP	12
1.5	Example of permutation chromosomes for the JSP	14
2.1	A Gantt chart which shows one of feasible schedule the instance shown in Table 2.1	22
2.2	Disjunctive representation of the instance shown in Table 2.1	25
2.3	A feasible schedule for the instance shown in Table 2.1 and Fig. 2.2	25
2.4	Example of disjunctive graph representation for the FJSP	29
2.5	Example of precedence vectors and their crossover procedure	31
3.1	Classification example of operations by the time window of $[2, 8]$	40
3.2	Comparison of decreasing speed of tardiness for benchmark instances with $f = 1.3$	49
4.1	Overview of the proposed method	60
4.2	Comparison of decreasing speed of makespan for DP data	68
4.3	Comparison of decreasing speed of makespan for DPdata in long-term	74
4.4	Comparison of decreasing speed of makespan for newly generated problems	74

List of Tables

1.1	Example of 3×3 problem	8
2.1	An instance of JSP with 3 jobs and 3 machines	22
3.1	Comparison results of performance of the priority rule-based reconstruction added to iterated local search (LA01-LA20)	47
3.2	Comparison results of performance of the priority rule-based reconstruction added to iterated local search (LA21-LA40)	48
3.3	Comparison results of the proposed method with GLS on 10×10 benchmark instances	51
3.4	Comparison results of the proposed method with GLS on medium sized benchmark instances (LA01 - LA20)	53
3.5	Comparison results of the proposed method with GLS on medium sized benchmark instances (LA21 - LA40)	54
4.1	Number of problems where PRC performed better / worse than HGA. . . .	66
4.2	Performance comparison for DP data	67
4.3	Performance comparison for DP data in long-term	72
4.4	Performance comparison for newly generated problems	73

Chapter 1

Introduction

1.1 Background and Research Objective

The job-shop scheduling problem (JSP) is one of the most important combinatorial optimization problem whose objective is to find an optimal assignment of machines where multiple jobs must be concurrently processed with limited available machines. The jobs consist of a series operations, which is processed with a specific machine by exclusively occupying the machine for a specified amount of time. Additionally, each job has a specified technological order that restricts the processing sequence of the corresponding operations. In the classical JSP, the jobs are required to be processed on every machine once, meaning that each job has the same number of operations as the number of available machines.

Since the JSP is closely related to wide fields of scheduling problem in real environments such as logistics and manufacturing, many studies have been conducted to find efficient heuristic algorithms for solving the JSP. The JSP is classified to NP-hard problems and thus it is widely agreed that obtaining an optimal solution for the JSP found in practical environments where the number of jobs and machines can be extremely large is difficult. Therefore, approximation algorithms which try to find an efficient solution for the JSP within a reasonable time with no guarantee of the optimality has been actively studied.

Most early heuristic approaches for the JSP employ priority rule, which decides processing sequence of jobs on each machine with a predetermined rule when multiple jobs are ready to be processed on the machine at the same time of the schedule. They use an algorithm by Giffler, B. and Thompson, G. [GT60] that enumerates an active schedule, which is a subset of the feasible schedule for the JSP that is proven to contain at least one of the optimal schedules. Then, one of the schedule that complies with a priority rule is chosen among the enumerated active schedules. Therefore, the efficiency of the priority rule is evaluated by the average quality of an obtained schedule for various problems of the JSP. To achieve a high level of efficiency, priority rules extract the information on the job-shop environment with empirically selected indexes such as processing time of the corresponding operations for each job, the time when a machine completes any preceding operations, and the time when the machine is first available to the job. In early studies, various indexes are proposed and the priority rules are determined by arbitrarily selecting and combining these indexes. Most of those priority rules are classified by Panwalkar, S. and Iskander, W. according to the types of the information gathered by the indexes and the numbers of the indexes combined [PI77]. Although their simplicity and flexibility make it easy to apply priority rules in a real-world environment, their myopic feature of choosing only one solution based on a predetermined rule often leads to inefficient schedules compared to more sophisticated approaches which involve meta-heuristic algorithms.

Various meta-heuristic approaches such as local search and genetic algorithms have been attempted on the JSP and most of them are reported to be quite efficient. The most primitive neighborhood structure in local search is defined by exchanging a processing sequence between pairs of successively processed operations and simulated annealing [vLAL92] or tabu search [Tai94] is performed to prevent the search from stagnating on local optima. Succeeding studies have mainly focused on reducing computational efforts to investigate increased number of solutions. For example, Nowicki, E. and Smutnicki, C. proposed an efficient restriction to the neighborhood structure [NS96] while

Basal, E. and Vazacopoulos, A. developed a novel method of estimating the cost of the neighborhood solutions within significantly shorter time [BV98]. Watson et al. revealed that efficient algorithm for the JSP that is based on local search approaches includes both intensification and diversification process [pWHW05]. Intensification process is performed by traditional local searches where better solution is found by investigating around previously found high-quality solutions. In contrary, diversification process which is only included in more sophisticated procedures tries to generate an efficient initial solution for another iteration of local search by generally combining multiple high-quality solutions found by previous iterations of local search procedures. While local search procedures mainly aim at providing an efficient intensification process, genetic algorithms are focused on diversification process. To apply genetic algorithm to the JSP effectively, an efficient solution representation method is required. Falkenauer, E. and Bouffouix, S. proposed a machine-based representation by specifying a preference list of operations on each machine which is then used for selecting one solution from the active schedules [FB91]. Ryohei, N. and Takeshi, Y. employed a binary representation which defines a precedence relationship between every two operations on the same machine and proposed harmonization algorithms which repairs conflicting relationship to avoid generating unfeasible solutions [NY91]. Bierwirth, C. developed a permutation representation which is a preference list that contains each job number multiple times [Bie95]. The job numbers in the list is then translated into operations, whose relative position in the list shows a preference of the corresponding operation in the solution. The permutation representation also requires dedicated crossover and mutation to avoid generating an invalid chromosome. Among the three representations above, the permutation representation is most widely used because of the facts that it needs essentially one list to preserve the solution contrary to the machine-based representation and that its length is considerably shorter than that of the binary representation for the same size of problem. The combination of the local search heuristics and genetic algorithms realizes both intensification and diversification and

thus turned out to be further improve the solution of genetic algorithms. In fact, machine-based representation, binary representation, and permutation representation are all combined with local search procedure to improve the solution represented by each chromosome and the results shows their performance are significantly improved [QW12, HSEC09, GZZL11].

Meta-heuristics are also used to acquire an efficient priority rule for the JSP. One of the most serious drawbacks of the classic studies of a priority rule for the JSP is that its indexes are empirically determined by researchers. Therefore, switching multiple priority rules according to the status of job-shop environment can leads to better solution if appropriate rules are selected. To achieve this, Ulrich, D. and Erwin, P. provide a list of priority rules used in the decision process of the solution for a specific problem and optimized elements of the list by using genetic algorithm [DP95]. El-Bouri, A. and Shah, P. had a different approach of constructing an artificial neural network that receives the current status of job-shop environment and determines an appropriate rule [EBS06]. Connection weights of the network is then trained by sets of randomly generated problems and preliminarily calculated optimal solutions. Additionally, it is also possible that to obtain an optimized combination of the indexes for a priority rules by the meta-heuristics. Su et al. employed genetic programming where priority rules are represented by program trees with nodes of the indexes and arithmetic operations to combine them [NZJT13]. The program tree is optimized to obtain better solutions for specified problems to acquire an efficient priority rule. In contrast, Tamura et al. evolved priority rules to acquire a better solution to one problem, revealing that problems with the same size can require different priority rules for each problem to get an optimal result [TYSF13]. Even though their priority rule representation is much simpler than the program trees to acquire a specialized priority rule for each problem within reasonable time, each obtained priority rule is quite efficient on the intended problem. Those meta-heuristic approaches for priority rules can be divided into two classes. First, studies by Ulrich, D. et al. and Tamura et al. proposes a method to obtain specialized priority rules

for any given problems. Other researches above tries to acquire one general priority rule that is efficient for every problem. Although the former generally yields a better solution for various problems with no surprise, the latter is more flexible in that the obtained rule is used for every problem. However, in terms of solution quality, both of them are considerably inferior to meta-heuristics approaches which directly optimizes solutions for a given problem. Additionally, with more computational resources being available recently, direct meta-heuristic approaches can obtain better solutions within reasonable time. Therefore, those meta-heuristic approaches has not always been focused on as the approximation algorithms for the JSP.

As the basic heuristic algorithms for the JSP had been established, extended JSPs with additional constraints and objective function to represent real-world problems more accurately began to attract more and more researchers' attention. For example, semiconductor manufacturing operations can be represented by the extended JSP and optimized for improved productivity because diverse products of semiconductor chips are manufactured in a single facility by using very expensive machines [MFDP⁺11]. However, additional constraints and different criteria of the extended JSP requires another consideration of the approaches for the classic JSP in order to efficiently solve the practical problems [ZBH05]. Therefore, JSP with tardiness minimization and Flexible JSP are formulated and heuristic approaches for them have been developed. In the tardiness minimization JSP (TJSP), each job has different due date and importance weight that represents the penalty of completing the job behind its due date. Tardiness of a solution is calculated by accumulating weighted tardiness of the jobs that are completed behind their due date and the objective of the TJSP is to find a solution with minimum tardiness. Currently, most efficient meta-heuristics for the TJSP include local search procedures or genetic algorithms derived from that for the classic JSP. However, some of the neighborhood reduction techniques for the classic JSP are inefficient or even invalid for the TJSP. Therefore, relatively simple local search heuristics of iterated hill-climb local search with random movements

to escape from local optima are considered to be most successful within a reasonable computational time [Kre00]. This local search procedure is also combined with genetic algorithm to form one of the most efficient algorithms for the TJSP [EMDP08]. For priority rule-based approaches, the TJSP is interesting in that additional indexes must be included in efficient priority rules for the TJSP to extract information of the risk of violating due date. Priority rules with such indexes are composed mainly of slack time to the due date and the importance weight of each job and their efficiency are revealed by Vepsäläinen, A. and Morton, T. [VM87]. In the Flexible JSP (FJSP), operations must be assigned on one of a given set of available machines which differs from one operation to another contrary to the classic JSP where each operation is processed on one given machine. Additionally, processing time of an operation may differ depending on a machine to process the operation. This considerably expands the number of possible solution compared to the classic JSP and requires extended neighborhood structures for local search procedures. Although the basic meta-heuristics approaches for the classic JSP can be applied to the FJSP with minimal changes, the difference between them invalidates most of specialized techniques for efficient searches on the JSP in a similar way to that on the TJSP. Therefore, although tabu searches have been attempted on the FJSP [Bra93], genetic algorithms with additional chromosome to determine machine assignments [PMC08] and combination of the genetic algorithm with the simple hill-climbing local search procedure [GSG08] are more efficient on the FJSP. Priority rules can also be applied to FJSP with an additional procedure to assign machines to each operation by Nhu et al. [HTL07] and thus genetic programming approach to obtain an optimized rule are proposed [TH08].

This thesis focuses on developing effective heuristic algorithms for two extended JSP, namely the TJSP and the FJSP. Priority rule optimization by meta-heuristics is integrated in existing meta-heuristics approaches with local search procedures for them because their relatively more complicated constraints and objective functions than that of the classic JSP make local search

techniques for the classic JSP that can considerably reduce the computational efforts mostly inappropriate for them. Since the TJSP and the FJSP have different extensions from the classic JSP, however, it is implied that they have different features and thus different priority rule-based heuristics should be applied to each of them. Two main focuses of this thesis on approximation algorithms for scheduling problems are followings. First, the efficiency of the priority rule-based heuristics on the TJSP and the FJSP is investigated by integrating state-of-the-art heuristics methods with them and by comparing their performance. Second, it is expected that different natures of these two problems needs different application of the priority rule to each of them to get an optimal performance. For the TJSP, it is known that priority rules can gather additional information in the form of slack time to the due date. Therefore, it is assumed that priority rules are more efficient on the TJSP with that information and thus priority rules are used to improve an solution acquired through the existing local search procedure. On the other hand, the FJSP requires additional machine assignments to obtain a feasible solution and thus number of the possible solutions of the FJSP tends to be larger than that of the classic JSP with the same number of operations. To tackle this problem, initial solutions for a local search procedure must be carefully limited to a promising region of the solution space. In addition to the existing methods where initial solutions for the local search procedure is selected by using genetic algorithms, priority rules are separately evolved by genetic algorithm for each problem to acquire more efficient initial solutions.

1.2 Job-shop Scheduling Problem (JSP)

1.2.1 Description of the Problem

The JSP is described by a set of multiple jobs $J_1, J_2, \dots, J_n$ and a set of multiple machines $M_1, M_2, \dots, M_m$. Each job J_i is composed of a set of operations $\{o_{i,1}, o_{i,2}, \dots, o_{i,m}\}$, where $o_{i,j}$ denotes the j -th operations that belongs to job J_i . Operations of the same jobs must follow technological order, which

Table 1.1: Example of 3×3 problem

	Operation $o_{i,j}(r_{i,j}, p_{i,j})$		
	First	Second	Third
J_1	$(M_1, 2)$	$(M_3, 3)$	$(M_2, 4)$
J_2	$(M_1, 3)$	$(M_2, 2)$	$(M_3, 4)$
J_3	$(M_3, 1)$	$(M_1, 3)$	$(M_2, 2)$

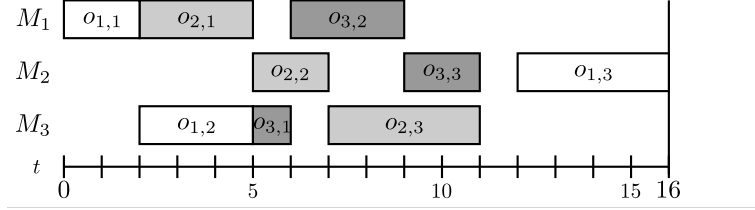


Figure 1.1: Gantt chart of an example schedule

means $o_{i,j}$ must be completed before $o_{i,j+1}$ is begun to be processed. Each operation is processed on a given machine by occupying the machine for a specified amount of time. Processing machine and time for an operation $o_{i,j}$ is respectively denoted by $r_{i,j}$ or $t_{i,j}$. Since each job needs to be processed on every machine once, the jobs are composed of m operations. Table 1.1 shows an example of 3 jobs and 3 machines problem.

Solutions of the JSP is described by specifying starting time $s_{i,j}$ for each operation $o_{i,j}$. An solution denoted by $\{s_{o_{1,1}}, s_{o_{1,2}}, \dots, s_{o_{1,n}}, s_{o_{2,1}}, \dots, s_{o_{n,m}}\}$ is feasible if and only if the following constraints are satisfied.

1. For any operation, $s_{o_{i,j}} \geq 0$ is satisfied
2. For any pair of operations $o_{i,j}, o_{i,k}$ ($k > j$) which belongs to the same job, $s_{o_{i,k}} \geq s_{o_{i,j}} + p_{o_{i,j}}$ is satisfied
3. For any pair of operations $o_{p,q}, o_{x,y}$ ($r_{p,q} = r_{x,y}$) which is processed on the same machine, either $s_{p,q} \geq s_{x,y} + p_{x,y}$ or $s_{x,y} \geq s_{p,q} + p_{p,q}$ is satisfied.

The first constraints shows that starting time of all operation is either zero or positive value. The second constraints ensures that the technological order is held and the third constraints states that machines cannot process more than one operation at the same time. One of possible feasible solu-

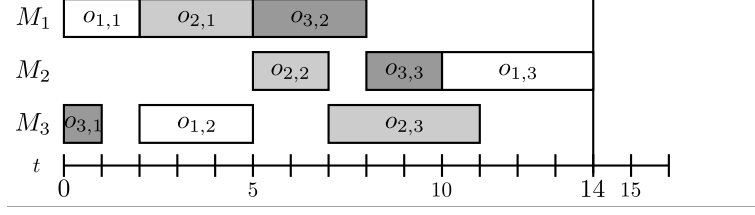


Figure 1.2: Example of an active schedule

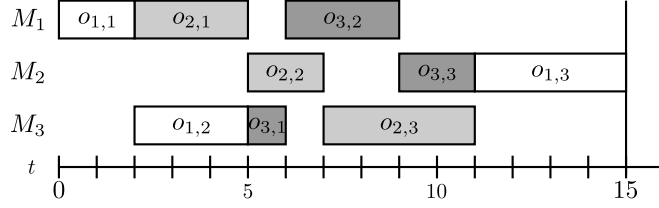


Figure 1.3: Example of a semi-active schedule

tions for the problem given by Table 1.1 is represented by $\{s_{o_{1,1}} = 0, s_{o_{1,2}} = 2, s_{o_{1,3}} = 10, s_{o_{2,1}} = 2, s_{o_{2,2}} = 5, s_{o_{2,3}} = 7, s_{o_{3,1}} = 3, s_{o_{3,2}} = 5, s_{o_{3,3}} = 8\}$. The feasible solution can also be represented by a gantt chart shown in Figure 1.1, where horizontal axis and vertical axis respectively shows time steps or machines. The objective of the JSP is to find a solution with minimal makespan $\max_{i,j} (s_{o_{i,j}} + p_{i,j})$.

It is widely known that a set of feasible solutions of the JSP contains two subsets, semi-active schedules and active schedules, where a set of active schedules is a subset of semi-active schedules. Both subsets are proven to contain at least one optimal solution and thus most approximation algorithms limits their searches to either active schedules or semi-active schedules.

Semi-active schedule meet the constraints of the JSP and no operation in the schedule can be processed earlier without changing operation sequence on the machine. For example, a solution shown by Figure 1.1 is not one of semi-active schedules because $o_{3,3}$ can be started earlier to obtain a semi-active schedule shown by Figure 1.3. Active schedules are feasible and no operations in the schedule can be started earlier without delaying other operations. Therefore, the solution shown by Figure 1.3 is not one of active schedules as $o_{3,1}$ can be processed earlier to acquire an active schedule shown by Figure

1.2.

1.2.2 Approximation Algorithms

Priority rules

Priority rule for the JSP obtain a solution by selecting one active schedule from the set of active schedules enumerated by GT algorithm [GT60]. First, GT algorithm assume a solution where multiple operations are processed on the same machine at the same time and where only technological order is met. Then, conflicting sets of operations that occupy the same machine at the same moment are identified. To correct the violations of the constraint, one of the conflicting sets that occurs earliest is chosen and one preceding operation is selected from the set to delay other operations in the set. This is repeated until all violation is corrected. Note that when different operation is selected from the conflicting set, elements of subsequent conflicting sets can be differ according to the selected operation due to the fact that delays are added to different set of operations. While GT algorithm lists every possible sequences of resolving conflict to enumerate active schedules, priority rules select one operation from every conflicting set by rules such as the followings.

FCFS (first come, first serve)

An operation that firstly become available on the machine is selected. First operation of each job is available on the corresponding machine at the time step of 0. Other operations become available on the machine as soon as a preceding operation of the same job is completed.

SPT (shortest processing time)

An operation with the shortest processing time on the machine is selected.

LPT (longest processing time)

An operation with the longest processing time on the machine is selected.

LRPT (longest remaining processing time)

An operation with the longest remaining processing time is selected,

where the remaining processing time is the sum of the processing time of operations which belongs to the same job and is succeeding in the technological order.

More complicated priority rules are composed of multiple rules. For example, indexes used in the rules above such as idling time of jobs, processing time of conflicting operations, and remaining processing time of jobs can be combined as a linearly weighted sum to provide a priority value which can be used to decide a preferred operation to resolve conflicts. Most early studies on priority rules are dedicated to empirically choosing indexes and combining them. Those indexes and composition are classified by Panwalkar, S. and Iskander, W. [PI77].

Recent studies focus on obtaining an optimized combination of the rules. These studies can be generally classified into two approaches. One tries to acquire a general rule which can yield good solutions for various problem and the other searches for a specific rule by performing independent optimization process on each problem. In the former approach, relatively long computational time for obtaining an optimized rule is accepted and thus the rule is represented by complicated mechanism such as programming trees [NZJT13] and artificial neural networks[EBS06]. These rules are optimized by a meta-heuristic algorithm or trained by previously acquired optimal solution for a select problems, respectively. The latter approach usually employs simpler representation like linearly weighted sum [TYSF13] or lists of primitive rules each of which is used for solving one conflicts [DP95]. The latter approach can also be seen as a meta-heuristic approach for the JSP where each solution is represented by a corresponding rule that select the solution from active solutions. From this perspective, priority rules restricts search space of the meta-heuristic approach in the latter approach because a set of solutions that can be represented by priority rules is usually a subset of active schedules.

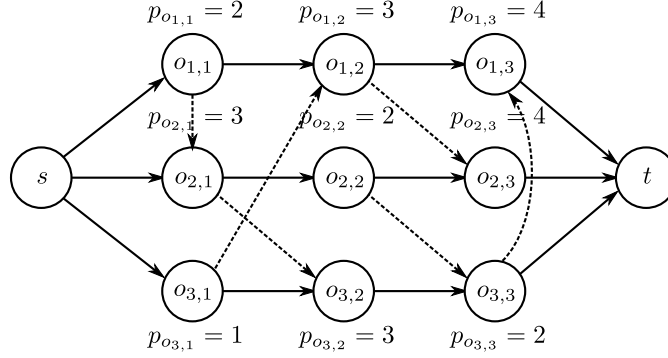


Figure 1.4: Example of a disjunctive graph for the JSP

Local search

Local search procedures for the JSP generally covers semi-active schedules to find an efficient solution. They require neighborhood structures so that neighborhood solutions is defined for any semi-active schedule as a subset of semi-active schedules. Starting from a given solution, makespans of neighborhood solutions of the current solutions are investigated and the current solutions are iteratively replaced by the neighborhood solution with minimum makespan. Additionally, most efficient local search procedures such as tabu search and simulated annealing provide mechanisms to escape from local optima where all of the neighborhood solutions have greater makespan.

To define effective neighborhood structure, solutions are generally represented by disjunctive graphs, which can represent any semi-active schedule. The disjunctive graph shown by Figure 1.4 corresponds to the solution shown by 1.3 Nodes and directed arcs of the disjunctive graph respectively denotes operations and precedence relationships and each node has weight value that denotes processing time required by the corresponding operation. Additionally, a source node s and a sink node t which are preceding or succeeding any operations are placed. Solid arcs in Figure 1.4 represents technological orders of each job and thus are shared on every solution for the same problem. Dotted arcs in the graph are placed between nodes which are processed on the

same machine to meet the constraint that machines cannot process more than one operations at the same moment. The disjunctive graph corresponds to a feasible solution if and only if all operations that are processed on the same machine are connected by a series of dotted arcs and the graph is acyclic. For example, in Figure 1.4, solid arcs between $o_{1,1}, o_{1,2}, o_{1,3}$ means these operations belong to the same job J_1 and dotted arcs between $o_{1,1}, o_{2,1}, o_{3,2}$ shows that they are processed in this order on machine M_1 . Therefore, the length of the longest path from a source node to any node of a valid disjunctive graph is equal to the time when the corresponding operation is begun to be processed on the corresponding semi-active schedule. Makespan of a solution given by valid disjunctive graph is also calculated by the longest distance from its source node to the sink node.

The longest path from a source node to a sink node in a disjunctive graph is called a critical path, which have interesting feature that any other disjunctive graph that shares all arcs included in the critical path cannot have shorter makespan. Therefore, effective neighborhood structure for the JSP excludes solution that preserves critical paths of the current solution. This is first revealed by van Laarhoven et al. [vLAL92] and most subsequent studies adopt disjunctive graph representation. Currently, tabu search that stores memories of previously investigated solutions and extracts the memories to avoid revisiting these solutions are considered to be among the most efficient approximation algorithms for the JSP. An early study on tabu search for the JSP defined neighborhood by exchanging processing sequencing between any adjacent operations in a critical path [Tai94]. However, it turned out that exchanging sequences between two operations does not lead to immediate makespan improvement as long as both of them does not connected to solid arcs that are included in the critical path. Therefore, more restricting neighborhood that excludes these exchanges are proposed [NS96]. Another remarkable study develops a method can accurately estimate makespans of neighborhood solutions with significantly less computational effort [BV98]. This enables tabu searches with wider neighbor within reasonable computa-

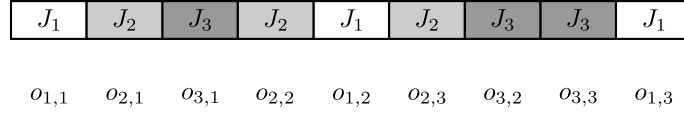


Figure 1.5: Example of permutation chromosomes for the JSP

tional time [ZLGR07].

Genetic algorithm

To apply genetic algorithms to the JSP, solutions must be represented as chromosomes and genetic operations such as crossover and mutation for the chromosomes must be defined. Chromosome representation methods for the JSP is classified to three types; machine-based representation, binary representation, and permutation representation. Since permutation representation is considered to be most flexible and widely used for both the classic JSP and extended JSPs, permutation representation is particularly discussed in this section. Machine-based representation consists of preference lists of the operation for each machine [FB91]. Each list contains every jobs once and each element corresponds to an operation which belongs to the job and is processed on the machine. When a chromosome is decoded to obtain a corresponding solution, an active schedule is generated in the similar method as priority rules. However, instead of priority rules, conflicts are resolved with the preference list by selecting the leftmost operation. This representation is inflexible in that it has multiple lists for a single chromosome and thus rarely used for the extended JSPs. Binary representation contains a list of binary values which defines precedence relationships between every pair of operations processed on the same machine [NY91]. To decode a chromosome that contains invalid precedence relationships such as cycles, harmonization algorithms are also proposed. However, this representation is incompetent to large scale problems because the length of the binary list grows rapidly as the number of jobs and machines are increased.

Permutation representation for the JSP with n jobs and m machines is composed of a list of $n \times m$ elements. In the list, each of the jobs $J_1, J_2, \dots, J_n$ is included m times. The list is scanned from left to right and k -th appeared J_i corresponds to the operation $o_{i,j}$. For example, Figure 1.5 shows a chromosome for a 3 jobs and 3 machine problem and a corresponding operation for each element. The chromosome is usually decoded into an active schedule by treating the chromosome as a preference list where leftmost operations are selected for resolving conflicts. The chromosome shown in Figure 1.5 is actually decoded into the schedule shown in Figure 1.3 and Figure 1.4. This is confirmed by the fact that $o_{1,1}, o_{2,1}, o_{3,2}$ are placed in this order in the chromosome which is consistent in the schedule.

Genetic algorithms can also be combined with local search heuristics. For example, Gao et al. apply a simple hill-climbing procedure to each chromosome to obtain superior results.

1.3 Outline of the Thesis

This thesis is composed of 5 chapters. In chapter 2, two extended JSPs focused on this thesis, namely TJSP and FJSP, is described and their formulation and existing heuristic algorithms for them is given. Then their different features are discussed and it is argued that appropriate approaches to find efficient algorithms for them is different. In chapter 3 and chapter 4, efficient heuristic algorithms for the extended JSPs are proposed by integrating priority rules with existing algorithms and their competence on benchmark problems are confirmed. The TJSP are tackled in the chapter 3 and the FJSP are dealt with in the chapter 4. Finally, this thesis is summarized in chapter 5.

1.4 Conclusion

In this chapter, research background and objective of this thesis is introduced. The JSP is formulated and existing heuristic approaches are described. In the following chapters, extended JSPs are described and efficient heuristic

algorithms for them is proposed.

Bibliography

- [Bie95] Christian Bierwirth. A generalized permutation approach to job shop scheduling with genetic algorithms. *Operations-Research-Spektrum*, 17(2-3):87–92, 1995.
- [Bra93] Paolo Brandimarte. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41(3):157–183, 1993.
- [BV98] Egon Balas and Alkis Vazacopoulos. Guided local search with shifting bottleneck for job shop scheduling. *Manage. Sci.*, 44(2):262–275, February 1998.
- [DP95] Ulrich Dorndorf and Erwin Pesch. Evolution based learning in a job shop scheduling environment. *Computers & Operations Research*, 22(1):25 – 40, 1995. Genetic Algorithms.
- [EBS06] Ahmed El-Bouri and Pramit Shah. A neural network for dispatching rule selection in a job shop. *The International Journal of Advanced Manufacturing Technology*, 31(3-4):342–349, 2006.
- [EMDP08] Imen Essafi, Yazid Mati, and Stéphane Dauzère-Pérès. A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. *Comput. Oper. Res.*, 35(8):2599–2616, August 2008.
- [FB91] E. Falkenauer and S. Bouffouix. A genetic algorithm for job shop. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 824–829 vol.1, Apr 1991.
- [GSG08] Jie Gao, Linyan Sun, and Mitsuo Gen. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers & Operations Research*,

- 35(9):2892 – 2907, 2008. Part Special Issue: Bio-inspired Methods in Combinatorial Optimization.
- [GT60] B. Giffler and G. L. Thompson. Algorithms for solving production-scheduling problems. *Operations Research*, 8(4):pp. 487–503, 1960.
- [GZZL11] Liang Gao, Guohui Zhang, Liping Zhang, and Xinyu Li. An efficient memetic algorithm for solving the job shop scheduling problem. *Computers & Industrial Engineering*, 60(4):699 – 705, 2011.
- [HSEC09] S.M.Kamrul Hasan, Ruhul Sarker, Daryl Essam, and David Cornforth. Memetic algorithms for solving job-shop scheduling problems. *Memetic Computing*, 1(1):69–83, 2009.
- [HTL07] Nhu Binh Ho, Joc Cing Tay, and Edmund M.-K. Lai. An effective architecture for learning and evolving flexible job-shop schedules. *European Journal of Operational Research*, 179(2):316 – 333, 2007.
- [Kre00] Stephan Kreipl. A large step random walk for minimizing total weighted tardiness in a job shop. *Journal of Scheduling*, 3(3):125–138, 2000.
- [MFDP⁺11] Lars Mönich, John W. Fowler, Stéphane Dauzère-Pérès, Scott J. Mason, and Oliver Rose. A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations. *Journal of Scheduling*, 14(6):583–599, 2011.
- [NS96] Eugeniusz Nowicki and Czeslaw Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42(6):797–813, 1996.

- [NY91] Ryohei Nakano and Takeshi Yamada. A genetic algorithm for job shop. In *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 474–479, Jan 1991.
- [NZJT13] Su Nguyen, Mengjie Zhang, M. Johnston, and Kay Chen Tan. A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem. *Evolutionary Computation, IEEE Transactions on*, 17(5):621–639, Oct 2013.
- [PI77] S. S. Panwalkar and Wafik Iskander. A survey of scheduling rules. *Operations Research*, 25(1):45–61, 1977.
- [PMC08] F. Pezzella, G. Morganti, and G. Ciaschetti. A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research*, 35(10):3202 – 3212, 2008. Part Special Issue: Search-based Software Engineering.
- [pWHW05] Jean paul Watson, Adele E. Howe, and L. Darrell Whitley. Deconstructing nowicki and smutnicki’s i-tsab tabu search algorithm for the job-shop scheduling problem. *Computers & Operations Research*, 33:2623–2644, 2005.
- [QW12] Ren Qing-dao-er-ji and Yuping Wang. A new hybrid genetic algorithm for job shop scheduling problem. *Computers & Operations Research*, 39(10):2291 – 2299, 2012.
- [Tai94] Eric D. Taillard. Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing*, 6(2):108–117, 1994.
- [TH08] Joc Cing Tay and Nhu Binh Ho. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering*, 54(3):453 – 473, 2008.

- [TYSF13] Yasumasa Tamura, Masahito Yamamoto, Ikuo Suzuki, and Masashi Furukawa. Acquisition of dispatching rules for job-shop scheduling problem by artificial neural networks using pso. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 17(5):731 – 738, 2013.
- [vLAL92] Peter J. M. van Laarhoven, Emile H. L. Aarts, and Jan Karel Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40(1):113–125, 1992.
- [VM87] Ari P. J. Vepsalainen and Thomas E. Morton. Priority rules for job shops with weighted tardiness costs. *Management Science*, 33(8):pp. 1035–1047, 1987.
- [ZBH05] Jeriad Zoghby, J. Wesley Barnes, and John J. Hasenbein. Modeling the reentrant job shop scheduling problem with setups for metaheuristic searches. *European Journal of Operational Research*, 167(2):336 – 348, 2005.
- [ZLGR07] ChaoYong Zhang, PeiGen Li, ZaiLin Guan, and YunQing Rao. A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. *Computers & Operations Research*, 34(11):3229 – 3242, 2007.

Chapter 2

Extended Job-shop Scheduling Problems

2.1 Introduction

The extended JSPs are problems with constraints and objective function slightly different from that of the classic JSP in order to better modelize real-world scheduling problems. Although their differences are so nominal that their generality to model various specific problems are not lost, it is enough to diminish the efficiency of the approximation algorithms specialized for the classic JSP. In this chapter, two main extended JSPs are described and heuristics approaches for them are presented and their difference are focused on.

Firstly, tardiness minimization JSP (TJSP) is described and heuristic approaches for the TJSP are presented. In the TJSP, due dates are added to each job and the objective function is replaced by minimization of total tardiness of all jobs. Since other constraints are retained, the numbers of possible solutions for the TJSP is the same as that for the JSP on problems with the same numbers of jobs and machines. The replaced objective function, however, generally increases the numbers of neighborhood solutions for local searches. Therefore, making efficient neighborhood structure for local searches in the TJSP is difficult. Secondly, flexible JSP (FJSP) and its approximation algo-

gorithms are explained. The FJSP has different constraints to the classic JSP in that each operation is processed on one of the available machines which are differently given for each machine. As each operation must be assigned to one of the multiple machines, this leads to increased number of possible solutions for the problems with the same numbers of machines and jobs. Additionally, the numbers of neighborhood solutions also tend to be greater. Finally, differences among the classic JSP, TJSP, and FJSP are discussed and proposal for more efficient approximation algorithms for the extended JSPs are explained.

2.2 Tardiness Minimization Job-shop Scheduling Problem

2.2.1 Description of the Problem

Job-shop environment with n jobs $J_1, J_2, \dots, J_n$ and m machines $M_1, M_2, \dots, M_m$ is considered. A job J_i ($1 \leq i \leq n$) consists of m operations $o_{i,1}, o_{i,2}, \dots, o_{i,m}$. An operation $o_{i,j}$ is processed on a machine $r_{i,j} \in \{M_1, M_2, \dots, M_m\}$ for processing time $p_{i,j} \geq 0$ without preemption. Each machine can process at most one operation at the same time and thus operations on the same machine must be processed in sequence. Each job must be processed on every machine exactly once. Thus, we assume that $r_{i,j} \neq r_{i,k}$ ($j \neq k \forall j, k$). Additionally, operations in the same job must be processed in the predetermined technological order. An operation $o_{i,j}$ is j -th operation in J_i and preceding operations $o_{i,1}, o_{i,2}, \dots, o_{i,j-1}$ must be finished before processing $o_{i,j}$. Additionally, due date d_i and weight w_i is specified for each job J_i in the tardiness minimization problem. An example with three jobs and three machines is shown in Table 2.1.

A schedule is specified by enumerating starting time $s_{i,j}$ for every operation $o_{i,j}$ and the schedule is feasible if and only if following constraints are satisfied.

1. For any operation, $s_{o_{i,j}} \geq 0$ is satisfied
2. For any pair of operations $o_{i,j}, o_{i,k}$ ($k > j$) which belongs to the same

Table 2.1: An instance of JSP with 3 jobs and 3 machines

Job	Weight	Due date	1st operation	2nd operation	3rd operation
J_1	1	14	$(M_1, 2)$	$(M_3, 3)$	$(M_2, 4)$
J_2	2	9	$(M_1, 3)$	$(M_2, 2)$	$(M_3, 4)$
J_3	4	12	$(M_3, 1)$	$(M_1, 3)$	$(M_2, 2)$

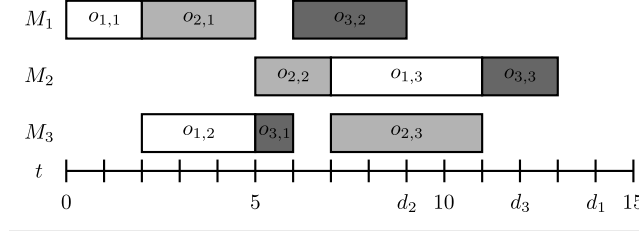


Figure 2.1: A Gantt chart which shows one of feasible schedule the instance shown in Table 2.1

job, $s_{o_{i,k}} \geq s_{o_{i,j}} + p_{o_{i,j}}$ is satisfied

- For any pair of operations $o_{p,q}, o_{x,y}$ ($r_{p,q} = r_{x,y}$) which is processed on the same machine, either $s_{p,q} \geq s_{x,y} + p_{x,y}$ or $s_{x,y} \geq s_{p,q} + p_{p,q}$ is satisfied.

The first constraint states that all jobs are released on time 0. The second and the third ensure that technological order is held for every job and each machine does not process multiple operations at the same time, respectively. Completion time C_i of job J_i is given by $C_i = s_{i,m} + p_{i,m}$ and J_i is tardy if $C_i > d_i$. One of feasible schedules for the instance shown in Table 2.1 is given by $\{s_{1,1} = 0, s_{1,2} = 2, s_{1,3} = 7, s_{2,1} = 2, s_{2,2} = 5, s_{2,3} = 7, s_{3,1} = 5, s_{3,2} = 6, s_{3,3} = 11\}$. Completion time of each job is $C_1 = 11, C_2 = 11$ and $C_3 = 13$. J_2 and J_3 are completed after their due dates and thus J_2 and J_3 are tardy jobs. This schedule is shown in Fig. 2.1 using Gantt chart where occupation of machines by each operation is presented with time as the horizontal axis.

The tardiness T_i for job J_i is defined by tardiness $T_i = \max\{C_i - d_i, 0\}$ with completion time and due date for the job. Therefore, only tardy jobs whose completion time is later than the due date affect the total weighted tardiness. Weight w_i represents importance of job J_i and higher value indicates that the delay of the job is heavily penalized. Overall, the objective function is the

minimization of the total weighted tardiness TWT shown by the following equation.

$$TWT = \sum_i w_i T_i \quad (2.1)$$

Note that in some cases it is impossible to complete all jobs in time depending on the tightness of due date and thus total weighted tardiness of optimal schedules differs by instances. The schedule shown in Fig. 2.1 has two tardy jobs J_2 and J_3 . Their tardiness values are $T_2 = 11 - 9$ and $T_3 = 13 - 12$ respectively and their weights are given by $w_2 = 2$ and $w_3 = 4$. Therefore, the total weighted tardiness of the schedule is $2 \times (11 - 9) + 4 \times (13 - 12) = 8$.

Similarly to the classic JSP, the set of all feasible schedules have three subsets, semi-active schedule, active schedule and non-delay schedule. Semi-active schedules are a set of schedules whose operations cannot be processed at an earlier time without modifying the sequence of the operations on each machine or breaking the constraints of schedule feasibility. The schedule shown in Fig. 2.1 is included in semi-active schedule. Active schedules are a set of schedules where each operation cannot be processed at an earlier time without delaying other operations or breaking the constraints. The schedule shown in Fig. 2.1 is excluded from active schedule because operation $o_{3,1}$ could be processed on $t = 0$ without delaying $o_{1,2}$ by exchanging the operation sequence on machine M_3 . In Non-delay schedule, no machine is kept idle when it could start processing some operation. The schedule in Fig. 2.1 is not a non-delay schedule because M_3 are kept idle on $t = 0$ although $o_{3,1}$ could be processed. Since active schedule is clearly a subset of semi-active schedule and it is proved that active schedule includes one of the optimal schedules, most methods search for an optimal schedule from a set of active schedule. GT algorithm proposed by [GT60] can enumerate any active schedule and thus GT algorithm is employed in methods whose solution space consists of active schedule. In the local search procedure, however, semi-active schedule is considered because the neighborhood solutions consist of semi-active schedule can be obtained much faster.

2.2.2 Approximation Algorithms

The local search procedures which are quite effective for the classic JSP can be performed on the TJSP with some adjustment to the neighborhood structures. For the makespan minimization of the classic JSP, disjunctive graph representation is used to restrict the neighborhood so that solutions which are proven not to improve the current solution are excluded from the neighborhood [Tai94]. This is realized by identifying operations that affect the completion time of the latest completed operation and more sophisticated neighborhood which effectively limit the neighborhood further are considered among the best algorithms for the classic JSP [NS96, ZLGR07]. For the tardiness criterion, all tardy jobs affect the value of the objective function and thus slightly modified neighborhood structure is used [PS99]. [Kre00] developed a large-step random-walk (LSRW) algorithm based on a modified neighborhood structure for the TJSP which tries to escape from local optima by random moves. LSRW is proven to be very effective for tardiness minimization.

Genetic algorithms for the classic JSP can also be applied to the TJSP with replacement for a fitness evaluation function. Since constraints of the TJSP are unchanged from that of the classic JSP, chromosome representation methods for the classic JSP can be used on the TJSP with no change. Therefore, Essafi et al. apply the local search procedure above to every chromosome of genetic algorithm [EMDP08] using the same representation as that for the classic JSP, which is one of the best heuristics for JSP with tardiness objective so far.

In the following, adjustment to local search procedure required to solve the TJSP is presented. Disjunctive representation for the TJSP is presented and neighborhood structure for the TJSP is explained using the disjunctive graph.

Disjunctive graph representation

A job-shop scheduling problem is represented by a disjunctive graph $G = (V, A, E)$, which is different from a disjunctive graph for the classic JSP in that sink nodes are given for each job. The vertex set V consists of the

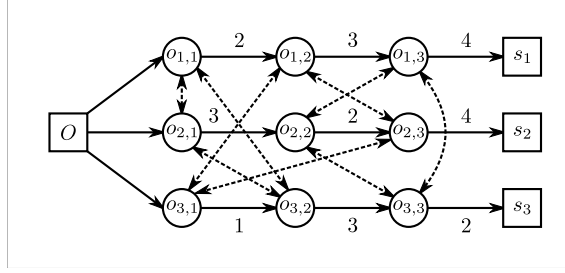
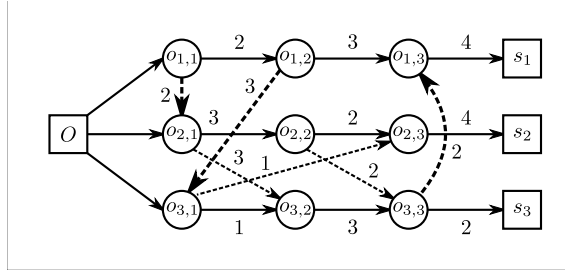
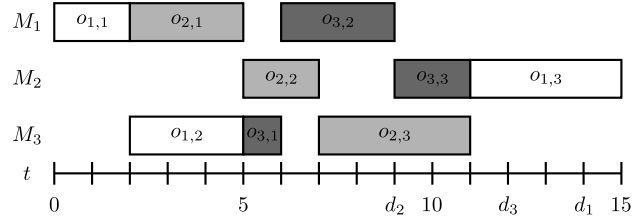


Figure 2.2: Disjunctive representation of the instance shown in Table 2.1



(a) Disjunctive graph representation of the feasible schedule



(b) Corresponding Gantt chart of the feasible schedule

Figure 2.3: A feasible schedule for the instance shown in Table 2.1 and Fig. 2.2

set of operations, dummy source nodes O and sink nodes for each job s_i ($i = 1, 2, \dots, n$). The directed arc set A represents technological order and connects consecutive operation in the same job. The disjunctive graph set E connects every pair of operations processed on the same machine. Directed arcs whose source node is $o_{i,j}$ has a weight of $p_{i,j}$. The problem instance shown in Table 2.1 is represented by the disjunctive graph in Fig. 2.2.

One of semi-active schedules is obtained by determining direction of each disjunctive graph. The direction represents the sequence in which operations are processed on the particular machine. Thus, this is equivalent to determine the processing order of operations on each machine. A corresponding schedule is feasible if and only if the graph is acyclic. Starting time of operation $o_{i,j}$

in the schedule is given by the length of the longest path from the source node O to the associated node of $o_{i,j}$. Therefore, a completion time of job J_i is the length of the longest path from O to a sink node s_i . In the classic JSP, one sink node is shared by all jobs, meaning that the longest path from the source node to the sink node is equivalent to the completion time of the latest job. However, in the TJSP, tardiness of each job must be calculated in order to evaluate a solution and thus the sink nodes are given for every job. A feasible schedule for the instance in Fig. 2.2 is shown in Fig. 2.3(a) and corresponding Gantt chart for the schedule is shown in Fig. 2.3(b). In this example, processing order on M_1, M_2, M_3 is given by $o_{1,1} \rightarrow o_{2,1} \rightarrow o_{3,2}$, $o_{2,2} \rightarrow o_{3,3} \rightarrow o_{1,3}$ and $o_{1,2} \rightarrow o_{3,1} \rightarrow o_{2,3}$ and completion time of jobs are $C_1 = 15, C_2 = 11, C_3 = 11$ respectively.

Neighborhood structure

The most common neighborhood structure for JSP is defined by an interchange of processing sequence on the same machine. For the classic JSP, the longest path from the source node to the sink node with the maximum completion time is defined as critical path and it is proven that interchanges of operations on a critical path always leads to a feasible schedule and that only such interchanges can reduce the makespan immediately [vLAL92]. Therefore typical neighborhood consists of interchanges of adjacent operations on a critical path.

For the tardiness minimization, however, completion time of all jobs can affect the objective function. Thus, critical paths are defined for each job by the longest path from the source node to the sink node. As reducing the completion time of on-time jobs does not lead to immediately improving the total weighted tardiness, neighborhood is defined by reversal of a disjunctive arc on the critical paths of tardy jobs. In the schedule shown in Fig. 2.3,

critical paths of J_1, J_2 and J_3 are following.

$$J_1 : O \rightarrow o_{1,1} \rightarrow o_{1,2} \rightarrow o_{3,1} \rightarrow o_{3,2} \rightarrow o_{3,3} \rightarrow o_{1,3} \rightarrow s_1$$

$$J_2 : O \rightarrow o_{1,1} \rightarrow o_{2,1} \rightarrow o_{2,2} \rightarrow o_{2,2} \rightarrow o_{2,3} \rightarrow s_2$$

$$J_3 : O \rightarrow o_{1,1} \rightarrow o_{1,2} \rightarrow o_{3,1} \rightarrow o_{3,2} \rightarrow o_{3,3} \rightarrow s_3$$

As completion time of J_1 and J_2 is later than due date, neighborhood is defined by a reversal of disjunctive arc $o_{1,2} \rightarrow o_{3,1}$, $o_{3,3} \rightarrow o_{1,3}$ or $o_{1,1} \rightarrow o_{2,1}$.

The neighborhood structure used by [Kre00] permits reversal of multiple disjunctive arcs and provides wider range of neighborhood. However, in the method by [EMDP08], neighborhood is limited to reversal of single arc because the local search procedure is combined with genetic algorithm and thus limited amount of computational effort should be dedicated to the local search procedure.

Since more critical paths can exist in the TJSP than in the classic JSP, the number of neighborhood solutions for the TJSP tends to be larger than that for the classic JSP. In the classic JSP, the latest operation of a job is included in critical paths if and only if the completion time of the job is one of the latest among all jobs. However, in the TJSP, any tardy job whose completion time is later than its due date generates a critical path. Since the possibility of multiple tardy jobs existing differentiate the TJSP from the classic JSP, there are multiple tardy jobs in solutions for almost all non-trivial problem of the TJSP. Therefore, local search procedures for the TJSP is not as efficient as that for the classic JSP due to the increased number of critical paths. For this reason, the approximation algorithms for the TJSP generally obtains a solution of worse quality compared to that for the classic JSP on problems with the same number of jobs and machines.

Iterative local search

If only improving moves are accepted with the neighborhood described above, the solution quickly reaches to one of local optima where no improving neighborhood is found. To continue local search process to find global optima,

moves with no improvement must be permitted to escape from the local optimum. The previous studies suggest that small numbers of successive random moves are sufficient to escape from local optima for the tardiness criterion [Kre00]. Although tabu search can also be used to prevent the convergence to local optima [Bon05], random move strategy with genetic algorithm [EMDP08] outperforms their results.

Priority rules

Priority rules for the classic JSP can also be applied. As due dates and weights are available for the TJSP. Several specialized rules for the TJSP are proposed. [VM87] compares existing priority rules with tardiness criterion and Apparent Tardiness Cost (ATC) rule performs well for various problems on average. ATC rule consists of two simple priority rules and combines them according to the situation. First, slack per remaining processing time (S/RPT) of each operation is calculated and operation with the least value is selected. Slack is the amount of the time that the job can be delayed without breaking the deadline when all subsequent operations are processed with no idle time. If slack of candidate operations is large enough, weighted shortest processing time rule which prioritizes an operation with shorter processing time and larger weight value is employed. Therefore, slack per remaining processing time and weighted shortest processing time are used in the linear combination.

2.3 Flexible Job-shop Scheduling Problem

2.3.1 Description of the Problem

In the FJSP, n jobs $J = \{J_1, J_2, \dots, J_n\}$ is processed on m machines $M = \{M_1, M_2, \dots, M_m\}$. A job J_i consists of n_i operations $(o_{i,1}, o_{i,2}, \dots, o_{i,n_i})$, which hold the technological sequence. Each operation $o_{i,k}$ must be processed on a set of machines $A_{i,k} \subseteq M$. Processing time on each machine may vary and that on machine M_j is denoted by $p_{i,k,j}$. The solution is represented by

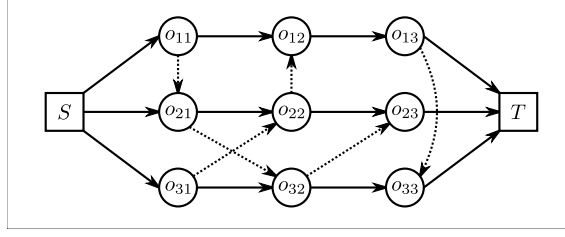


Figure 2.4: Example of disjunctive graph representation for the FJSP

selecting processing machine $M(o_{i,k}) \in A_{i,k}$ and specifying starting time $s_{i,k}$ for each operation $o_{i,k}$ and is feasible when all of the following constraints are satisfied:

1. Technological sequences are held, i.e., $s_{i,j} + p_{i,j,M(o_{ij})} \leq s_{i,k}$ is satisfied between any pairs of operations $j < k$ in the same job.
2. No two pairs of operations $o_{i,j}, o_{p,q}$ that are processed on the same machine ($M(o_{i,j}) = M(o_{p,q})$) is processed at the same time.

The FJSP's objective is to find a feasible solution that minimizes makespan $C_{\max}$ where C_i is completion time $s_{i,n_i} + p_{i,n_i,M(o_{in_i})}$ of each job J_i . This FJSP is more complex than the classic JSP in that $M(o_{i,k})$ must be selected for each operation $o_{i,k}$ and thus is classified as NP-hard. Various metaheuristics have been applied to the FJSP, tabu search and genetic algorithm are known to be most efficient. However, hybrid genetic algorithm (HGA) that applies a local search procedure to each chromosome of its population in addition to a traditional genetic operator is recently proposed and confirmed to have superior performance. Neighborhood structures and the encoding scheme adopted by these methods are summarized in the following.

2.3.2 Approximation Algorithms

Neighborhood structures for local search

The neighborhood structure is defined with disjunctive graph representation to perform substantial changes in the solution in a similar approach for the classic JSP. A disjunctive graph is represented in the same way for the classic

JSP as shown in Fig. 2.4, where nodes correspond to individual operations with dummy start and termination nodes denoted by S and T . Each node has a weight equal to the processing time for the operation. Arcs show the precedence of processing sequences. Solid arcs are connected between operations that belong to the same job and represent topological sequencing and dashed arcs that are processed on the same machine. Makespans of solutions are calculated by the longest path length between S and T . The earliest possible start time for individual operations is represented by the length of the longest path between S and each node. A disjunctive graph that is cyclic means that the corresponding solution is infeasible.

The neighborhood of a solution is defined by the set of acyclic disjunctive graphs acquired by deleting one node from the corresponding disjunctive graph and inserting it at another position. The difference from the neighborhood structures for the classic JSP is that the FJSP permit some operations to be processed on a different machine. Therefore, a deletion and insertion of these operations into another chain of dashed arcs should be considered as well as a reversal of one dashed arcs investigated in the classic JSP. It is proved in [MG00] that all neighborhood solutions with smaller makespans must have the following properties:

1. A node that is deleted and inserted is included in the longest path from S to T .
2. $o_{i,j}$ is inserted after each operation $o_{p,q}$ that satisfies $S_{p,q} + p_{p,q,M(o_{pq})} > T_{i,j}^-$ and $S_{p,q} + p_{p,q,M(o_{pq})} \leq S_{i,j}^-$ where $S_{i,j}$ and $T_{i,j}$ are the longest path length between S or T and $o_{i,j}$. '-' indicates a situation in which dashed arcs connected to $o_{i,j}$ are deleted.
3. $o_{i,j}$ is inserted before each operation $o_{p,q}$ that satisfies $S_{p,q} + p_{p,q,M(o_{pq})} \leq T_{i,j}^-$ and $S_{p,q} + p_{p,q,M(o_{pq})} > S_{i,j}^-$.

These properties help reduce the number of neighborhood solutions investigated and the computation time needed for local search procedures.

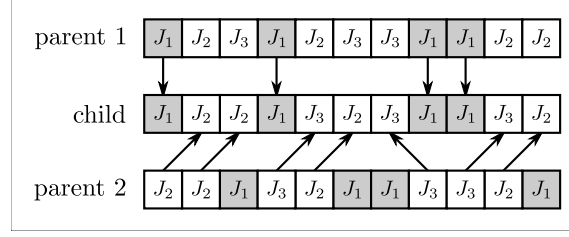


Figure 2.5: Example of precedence vectors and their crossover procedure

Genetic algorithm encoding scheme

The encoding scheme commonly used for the FJSP consists of two integer vectors whose length is equal to the total number of operations. The first vector is used to select a machine to be processed for each operation while the second vector which is essentially the same as the permutation representation for the classic JSP decides operation sequencing on each machine. Each element in the first vector defines the machine assignment for the corresponding operation. Elements of the second vector consist of job $J_i \in J$. J_i appears n_i times. Each element corresponds to an operation and j -th appearing J_i from the left corresponds to $o_{i,j}$. This vector defines the precedence between operations on the same machine. The leftmost operation is preferred if a conflict occurs.

An example of the first vector for an instance with $J = \{J_1, J_2, J_3\}$ and $n_1 = 4, n_2 = 4, n_3 = 3$ is as follows:

$$\{M_1, M_2, M_1, M_3, M_2, M_1, M_3, M_1, M_2, M_2, M_3\}$$

First element M_1 shows that $o_{1,1}$ is processed on M_1 and fifth element M_2 represents a machine where $o_{2,1}$ is processed. An example of the second vector for the above instance is as follows:

$$\{J_1, J_2, J_3, J_1, J_1, J_2, J_2, J_3, J_2, J_3, J_1\}$$

The second and the fifth elements correspond to $o_{2,1}, o_{1,3}$. Their sequence of appearance for the corresponding element means that $o_{2,1}$ is processed before $o_{1,3}$.

Although any of the usual crossover operators, such as two-point or uniform crossover, may be used, they often produce infeasible solutions for the sec-

ond vector which is why Precedence Preserving Order-based crossover (POX) [LYL98], shown in Fig. 2.5, is used. Elements corresponding to a randomly selected job are preserved and other elements are inherited from the other parent, preserving the precedence in the parent vector. Mutation of the first vector is defined by randomly changing elements. Mutation of the second vector is commonly implemented by exchanging two randomly selected elements.

Priority rules

Studies on priority rules for the FJSP mainly focuses on how to select a processing machine for each operation. This is because an assignment of a machine must be determined for each operation before processing sequencing on the same machine can be resolved using traditional priority rules for the classic JSP. Some genetic algorithms for the FJSP initialize its population using priority rules [PMC08, ZGS11]. They decide machine assignment for all operations prior to determining the processing sequence of operations on each machine so that the total processing time of operations assigned on each machine becomes equal. As a result, operations cannot be processed on machines other than the predetermined machine even if other machines are idle, leading to a low machine utilization of the resulting schedule. Learnable Genetic Architecture (LEGA) by Ho et al. [HTL07] thus solves the issue by generating populations by selecting processing machine which can complete the operation earliest and by determining processing sequence with three basic priority rules simultaneously. Tay and Ho also proposed a composite priority rule and its optimization by genetic programming using the least waiting assignment for machine selection and a tree-based composite priority rule for deciding the operation sequence [TH08]. The proposed construction procedure determines machine assignment and processing sequence simultaneously similar to LEGA and uses a combination of multiple basic priority rules as well as the genetic programming approach.

2.4 Conclusion

In this chapter, the TJSP and the FJSP are described and heuristic approaches for solving them are explained. The local search approaches that is very efficient on the classic JSP is not necessarily effective on the two extended JSPs. For the TJSP, any exchanges that affects the starting time of any operation that belongs to the jobs which violate the due date. Therefore, the number of the neighborhood solutions tend to be large, leading to a longer computation time for local search procedures. For the FJSP, as operations can be processed on different machines, the number of the neighborhood solutions involved by one operation is as large as the number of machines which are available to them. This also leads to a increased number of neighborhood solutions, making local search procedures inefficient on the FJSP. However, local search is still one of the most promising approaches for both the TJSP and the FJSP because solution quality by local search is still very competitive to that of other methods such as genetic algorithm on the extended JSPs.

The priority rules can enhance effectiveness of local search procedures for those extended JSPs. First, efficient priority rules of such as ATC are proposed for the TJSP using due dates and weight for each job which is exclusively available information on the TJSP. Since the number of possible solutions on the TJSP with the same number of jobs and machines are not increased from that on the classic JSP, priority rules are expected to be better used as a method to supplement intensification process of local search procedures. To achieve this, a priority rule-based procedure to effectively search for better solutions that are similar to particular solution is proposed, integrated to an existing local search heuristic, and evaluated in chapter 3. Second, local searches for the FJSP is less effective due to the fact that they need to search optimal solutions from greater number of possible solutions on the problem with the same number of jobs and machines. Therefore, priority rules can enhance local search procedures for the FJSP by providing promising solutions to start local search among the increased possible solutions. In chapter 4, combined priority rules

for the FJSP which determines both machine assignments of operations and operation sequencing on machines are proposed to obtain promising solutions by acquiring better composition of rules.

Bibliography

- [Bon05] K.M.J. Bontridder. Minimizing total weighted tardiness in a generalized job shop. *Journal of Scheduling*, 8(6):479–496, 2005.
- [EMDP08] Imen Essafi, Yazid Mati, and Stéphane Dauzère-Pérès. A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. *Comput. Oper. Res.*, 35(8):2599–2616, August 2008.
- [GT60] B. Giffler and G. L. Thompson. Algorithms for solving production-scheduling problems. *Operations Research*, 8(4):pp. 487–503, 1960.
- [HTL07] Nhu Binh Ho, Joc Cing Tay, and Edmund M.-K. Lai. An effective architecture for learning and evolving flexible job-shop schedules. *European Journal of Operational Research*, 179(2):316 – 333, 2007.
- [Kre00] Stephan Kreipl. A large step random walk for minimizing total weighted tardiness in a job shop. *Journal of Scheduling*, 3(3):125–138, 2000.
- [LYL98] Kyung-Mi Lee, T. Yamakawa, and Keon-Myung Lee. A genetic algorithm for general machine scheduling problems. In *Knowledge-Based Intelligent Electronic Systems, 1998. Proceedings KES '98. 1998 Second International Conference on*, volume 2, pages 60–66 vol.2, Apr 1998.
- [MG00] M. Mastrolilli and L. M Gambardella. Effective neighbourhood functions for the flexible job shop problem. *Journal of Scheduling*, 3(1):3–20, 2000.

- [NS96] Eugeniusz Nowicki and Czeslaw Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42(6):797–813, 1996.
- [PMC08] F. Pezzella, G. Morganti, and G. Ciaschetti. A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research*, 35(10):3202 – 3212, 2008. Part Special Issue: Search-based Software Engineering.
- [PS99] Michael Pinedo and Marcos Singer. A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop. *Naval Research Logistics (NRL)*, 46(1):1–17, 1999.
- [Tai94] Eric D. Taillard. Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing*, 6(2):108–117, 1994.
- [TH08] Joc Cing Tay and Nhu Binh Ho. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering*, 54(3):453 – 473, 2008.
- [vLAL92] Peter J. M. van Laarhoven, Emile H. L. Aarts, and Jan Karel Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40(1):113–125, 1992.
- [VM87] Ari P. J. Vepsalainen and Thomas E. Morton. Priority rules for job shops with weighted tardiness costs. *Management Science*, 33(8):pp. 1035–1047, 1987.
- [ZGS11] Guohui Zhang, Liang Gao, and Yang Shi. An effective genetic algorithm for the flexible job-shop scheduling problem. *Expert Systems with Applications*, 38(4):3563 – 3573, 2011.
- [ZLGR07] ChaoYong Zhang, PeiGen Li, ZaiLin Guan, and YunQing Rao. A tabu search algorithm with a new neighborhood structure for the

job shop scheduling problem. *Computers & Operations Research*,
34(11):3229 – 3242, 2007.

Chapter 3

Priority rule-based method for Tardiness Minimization

3.1 Introduction

In this section, the minimization of total weighted tardiness in the job-shop environment is considered. Due dates and weights are assigned to each job and tardiness penalty is given with respect to the weight if the completion time is later than the due date. There are roughly two kinds of approaches for the minimization of total weighted tardiness. First one is using iterated local search procedures proposed by [Kre00] and [EMDP08], which are quite efficient in terms of performance on standard benchmark problems [PS99]. The large step random walk proposed by [Kre00] consists of iterative improvement to intensify search and random walk to escape from the local optima. The genetic local search (GLS) proposed by [EMDP08] optimizes initial solutions for the iterative local search by genetic algorithm and is considered to be one of the most effective methods. They are also superior to the shifting bottleneck heuristic [PS99] which decomposes JSP into multiple sub-problems of determining assignments on a single machine and are considered to be ones of the most efficient approaches to JSP with total weighted tardiness criterion.

Another approach is based on priority rules that have been extensively studied especially for tardiness objective. Early works are reviewed by [PI77].

These priority rules determine the sequences of operations on machines according to various information about operations such as processing time, remaining operation number and due date. The goal of the studies is to develop an efficient priority rule to obtain better solutions in different problems and [VM87] confirmed that appropriate combination of processing time and due date leads to a priority rule which performs better than other priority rules on average. However, priority rules are generally less efficient compared to local search algorithms because the operation sequences are optimized for a particular problem instance in local search algorithms. Recently, [ZCL09] and [ZSW13] exploit priority rules to guide the search for promising schedule for a particular instance. [ZCL09] determine the first operations of the sequences by genetic algorithm while the rest of the sequences are determined with a priority rule. Thus, the search space of genetic algorithm is reduced by the priority rule to improve its computational efficiency. [ZSW13] extract scheduling policies from a feasible schedule in a form of priority rules and feasible schedules are generated from the policies to obtain a more efficient schedule. However, their performance is worse than state-of-the-art local search procedures on small benchmark problems and their solution representation is so different from the one used in local search algorithms that their methods have not been incorporated with efficient local search algorithms.

An efficient heuristic method for the minimization of total weighted tardiness is proposed by incorporating priority rule with local search methods. Conventionally, the whole schedule are determined with a priority rule. Therefore, priority rules have been used only to generate initial solutions of local search methods. In this chapter, rule-based reconstruction procedure within arbitrary selected time window is proposed to modify a feasible schedule with priority rules and to generate neighborhood schedules with priority rules during local search procedure. This rule-based reconstruction procedure defines a sub-problem for a feasible schedule and provides another feasible schedule by solving it with priority rules. The sub-problem is defined by a decision of machine assignments of jobs within an arbitrary selected time window

where other assignments are determined according to the feasible schedule. Since [TYSF13] shows that linearly-combined priority rules are more effective for makespan criterion if the gains are optimized for particular instance, the linear-combined priority rules are adopted in our rule-based reconstruction procedure. The time window and the gains of linear-combines priority rules are optimized by genetic algorithm because most of the combination provides a schedule with worse tardiness and we are interested only in improving the tardiness of the schedule. In the proposed method, local search procedure in GLS is adopted and rule-base reconstruction procedure above is employed to escape from local optima. As a result, the proposed method guides the conventional local search procedure towards more promising solutions by priority rules and thus the proposed method outperforms GLS which shows best performance so far.

3.2 Proposed Approximation Algorithm

The proposed method alternately applies conventional local search procedure and rule-based reconstruction procedure. The rule-based reconstruction procedure exploits priority rules to efficiently improve a feasible schedule obtained from the conventional local search procedure. If the current solution cannot be improved by the rule-based reconstruction procedure further, the conventional local search procedure is applied to improve the current solution. Typically, priority rules determine the whole operation sequences of the schedule and this prevents integration with local search methods as the sequences obtained by local search methods are completely overwritten by priority rules. Rule-based reconstruction procedure addresses this problem by selecting a time window and updating the operation sequences with priority rules only within the time window. Priority rules are given by linear combination of basic priority rules proposed by [TYSF13]. Selection of the time window and gains of priority rules is vital and should be dynamically optimized according to the given schedule. Therefore, they are optimized with genetic algorithm.

3.2.1 Rule-based reconstruction procedure

When a problem instance of JSP, a feasible schedule for the instance (denoted by the enumeration of starting time of operations $s_{i,j}$ ($i = 1, 2, \dots, n$, $j = 1, 2, \dots, m$)) and a time window $[t_s, t_e]$ ($t_s < t_e$) are given, the operations are divided into three categories P , L and S . If $s_{i,j} < t_s$ (i.e. starting time of operation $o_{i,j}$ is earlier than t_s), $o_{i,j}$ is included in P . Similarly, $o_{i,j}$ is classified to S if $s_{i,j} > t_e$. Otherwise, operation $o_{i,j}$ is an element of L . Fig. 3.1 shows an example with time window $[2, 8]$ where operations are classified to $P = \{o_{1,1}\}$, $L = \{o_{2,1}, o_{3,2}, o_{2,2}, o_{1,2}, o_{3,1}\}$ and $S = \{o_{3,3}, o_{1,3}\}$.

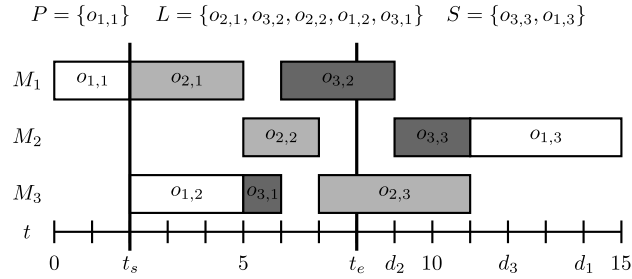


Figure 3.1: Classification example of operations by the time window of $[2, 8]$.

In priority rule-based reconstruction procedure, the schedule is reconstructed to one of active schedules. In the reconstructed schedule, the sequences of operations included in P or S are retained as much as possible while sequences of the operations in L is determined with priority rules. To obtain the reconstructed schedule, GT algorithm by [GT60] is employed to acquire an active schedule and conflicts between operations are resolved by previous starting time or priority rules depending on their class. The following procedure obtains the reconstructed schedule.

1. Let $A = \{o_{1,1}, o_{2,1}, \dots, o_{n,1}\}$ be the set of all candidate operations
2. Let $B = \emptyset$ be the set of all scheduled operations
3. Repeat step 4. to step 8. until starting time of all operations are determined and $A = \emptyset$.

4. Find the operation $o' \in A$ with the earliest possible completion time c' satisfying constraints with respect to any operation in B
5. Find the set of candidate operation C which is processed on the same machine as o' and the earliest possible starting time is less than c'
6. Choose the operation o'' from C . If no operation in C is classified to P and any operation in C is classified to L , the operation is selected by priority rules from all operations included in C and belong to L . Otherwise, the operation with the earliest possible starting time in the previous schedule is selected.
7. Starting time of o'' in the reconstructed schedule is given by the earliest possible starting time of o''
8. Add o'' to B . If o'' is not the final operation in the job, add the subsequent operation of o'' to A

3.2.2 Linear combination of priority rules

Existing priority rules for the TJSP with Apparent Tardiness Cost (ATC) [VM87] is used. On the other hand, [TYSF13] shows that linear combination of priority rules can be more effective than conventional priority rules if the gains are optimized for a particular instance with makespan criterion. The proposed method exploits priority rules to guide the local search procedure and thus a priority rule optimized for a particular instance is more useful than a priority rule which performs better in general. Therefore, linear combination of existing rules is used in the proposed method and its gains are optimized.

ATC rule consists of two simple priority rules and combines them according to the situation. Slack per remaining processing time (S/RPT) which is the ratio between processing time of remaining operations and permissible delay not to break the due date is calculated for each operation and one with the least value is selected. If slack of candidate operations is large enough, weighted shortest processing time (WPT) rule which prioritizes an operation

with shorter processing time and larger weight value is employed. ATC rule is typically used to generate a non-delay schedule. However, the proposed method searches for active schedule. As active schedules permit each machine to be idle waiting for a operation, possible machine idle time for a operation is considered so that an operation which leads to longer idle time can be penalized.

The priority rule used in the rule-based reconstruction procedure calculates priority value v_o for each candidate operation by the following equation and prioritizes the operation with minimum priority value.

$$v_o = \alpha_1 \text{WPT}_o + \alpha_2 \text{S/RPT}_o + \alpha_3 \text{PIM}_o \quad (3.1)$$

where α_1, α_2 and α_3 are gain parameters to be optimized and WPT, S/RPT and PIM are normalized weighted processing time, slack per remaining time and possible idling time of corresponding machine for each operation. For an operation $o_{i,j}$ whose earliest possible starting time is t_s and the machine to be processed becomes idle at time $pm_{o_{i,j}}$, these values are calculated with the following equations using the earliest possible starting time $t_{o_{i,j}}$ of the operation $o_{i,j}$ and the earliest possible completion time c' of o' in the reconstruction procedure described above.

$$\text{WPT}_{o_{i,j}} = \frac{1}{w_i \max_{i,j} p_{i,j}} \frac{p_{i,j}}{p_{i,j}} \quad (3.2)$$

$$\text{S/RPT}_{o_{i,j}} = \max \left\{ \min \left\{ \frac{d_i - t_{o_{i,j}} - \sum_{k=j+1}^m p_{i,k}}{\sum_{k=j+1}^m p_{i,k}}, 0.0 \right\}, 1.0 \right\} \quad (3.3)$$

$$\text{PTM}_{o_{i,j}} = \frac{c' - pm_{o_{i,j}}}{t_{o_{i,j}} - pm_{o_{i,j}}} \quad (3.4)$$

Note that slack per remaining time is truncated because most of efficient rules including ATC ignore negative slack value.

3.2.3 Integration with local search procedure

A feasible schedule obtained by the local search procedure can be reconstructed by the procedure described above if a time window and gains for the priority rule is given. Since the objective function is the minimization of

the total weighted tardiness, we search for a schedule with less total weighted tardiness by the reconstruction procedure. Thus, the time window and the gains which minimizes the total weighted tardiness is required for the reconstruction. However, it is difficult to enumerate possible values and calculate the total weighted tardiness of the reconstructed schedules. Therefore, a time window and gains are optimized by genetic algorithm. The chromosome of genetic algorithm is represented by five dimensional vector of real numbers $\{c, r, \beta_1, \beta_2, \beta_3\}$. Parameters c and r determine a time window in which operations are prioritized by the linear combined priority rule. The time window is given by $t_s = C_{\max}(c - r)$ and $t_e = C_{\max}(c + r)$ where $C_{\max}$ is a makespan of the current schedule. Gain parameters of the linear combined rule are given by $\alpha_1 = \beta_1 - 0.5, \alpha_2 = \beta_2 - 0.5, \alpha_3 = \beta_3 - 0.5$.

The current solution is initialized with a random schedule. Rule-based reconstruction procedure and the iterative local search procedure is alternately applied. In the rule-based reconstruction procedure, the current schedule is updated only when a parameter set of time window and gains which reduces the total weighted tardiness of the current schedule is found. If no improving parameter set is found after a certain number of generations $r_{\max}$, we assume that the current solution cannot be improved with rule-based reconstruction procedure and the iterative local search procedure is applied.

The iterative local search procedure is similar to the one used by [EMDP08]. In the iterative local search employed in the proposed method, however, the best solution found by the procedure is retained and next iteration of the iterative local search procedure is started from the best solution of the previous search. Therefore, solutions surrounding the best solution are more intensively searched when the iterated local search procedure is performed multiple times. Starting from an initial solution, improvement and perturbation are performed alternately. Improvement consists of steepest descent search where the most improving move in the neighborhood is selected. Once the improvement algorithm reaches to one of local optima, perturbation is performed by random move. A move is randomly selected from the neighbor-

hood regardless of whether it is improving or not. Random move is performed t times where t is chosen by a uniform random number $[t_{\min}, t_{\max}]$. As suggested by [EMDP08], these parameters are given by $t_{\min} = 5$ and $t_{\max} = 8$. The procedure stops if improvement procedure finds local optima n_{sd} times. The best solution with the minimum total weighted tardiness during the procedure is retained and the current solution is replaced by the best solution. However, if the best solution has the same total weighted tardiness as that of the initial solution, we assume that the best solution is the same as the initial solution and the current solution is replaced with the local optimum obtained by the last improvement to avoid premature convergence.

The overall procedure of the proposed method is the following.

1. Initialize the current solution with a random schedule
2. Initialize g chromosomes of five dimensional vectors with random uniform number $[0, 1]$
3. Set $r = r_{\max}$
4. If $r = 0$, go to step 8
5. Calculate the fitness value of each chromosome by performing rule-based reconstruction of the current schedule
6. If a schedule with less total weighted tardiness is found, update the current solution and set $r = r_{\max}$
7. Decrement r and go back to step 4.
8. Perform the iterative local search n_{sd} times starting from the current solution
9. Replace the current solution by the iterative local search procedure.
10. If terminal condition is met, stop the procedure. Otherwise, go back to step 3

Step 3. to 7. perform local search through rule-based reconstruction procedure. Time window and rule gains are selected by the chromosomes and fitness value of each chromosome is given by total weighted tardiness of the reconstructed schedule. If the reconstructed schedule has the same critical path as the previous schedule, the fitness value are doubled to penalize premature convergence. Crossover operation are defined by uniform crossover of each elements of the five dimensional vector while mutation operation randomly selects one element and changes it with uniform random number $[0, 1]$. If an improving schedule is found by the procedure above, the current schedule is updated. This local search with rule-based reconstruction proceeds until improving chromosome is not found in $r_{\max}$ successive generations. n_{sd} , g and $r_{\max}$ defines computational efforts dedicated to each procedure. The larger n_{sd} spends more time on the conventional local search procedure while larger g and $r_{\max}$ take more time to rule-based reconstruction procedure.

3.3 Performance Evaluation

In order to verify the effectiveness of the proposed method and the priority rule-based reconstruction procedure, computational experiments are performed. The proposed method is implemented and executed on a personal computer with 3.3GHz Core i5 processor and 8GB RAM.

Benchmark instances proposed by [PS99] and [EMDP08] are used to evaluate the quality of heuristic methods. These instances are derived from benchmark instances for makespan criterion with additional weights and due date to jobs. For 20 % of the total jobs, $w_i = 4$ is given and $w_i = 2$ is given for 60 % of the jobs. Remaining 20 % of the jobs have $w_i = 1$. Due date are given by $d_i = \lfloor f \sum_{j=1}^m p_{i,j} \rfloor$ for all jobs where f is a tightness factor of due dates. Three values of tightness factor $f = 1.3$, $f = 1.5$ and $f = 1.6$ are considered. Benchmark by [PS99] consists of 22 instances with 10 machines and 10 jobs. Some instances have larger number of jobs in the original benchmark set and last jobs are eliminated in this benchmark set. Optimal values

of total weighted tardiness for these instances are calculated with branch and bound method by [SP98]. However, better solution is found by subsequent research [EMDP08] and thus the branch and bound algorithm is considered to be stopped before finding the optimal solution. 40 benchmark instances proposed by [EMDP08] include larger instance with up to 300 operations. So far, optimal values of total weighted tardiness for these instances have not been known due to their larger size.

3.3.1 Performance of rule-based reconstruction procedure

To evaluate the performance of the priority rule-based reconstruction integrated to the existing iterative local search procedure, solution quality of the the iterative local search procedure and the iterative local search supplemented by priority rule-based reconstruction is compared. Additionally, results of the priority rule-based reconstruction procedure without time window is also compared to identify the effectiveness of partial reconstruction of solutions by time window to intensify its search on solutions which is similar to previously found good solutions. The number of chromosomes in rule-based reconstruction procedure is fixed at $g = 50$. As suggested by [EMDP08], the maximum iteration number of conventional local search procedure n_{sd} is given by a uniform random number $[n_{sd}^{\min}, n_{sd}^{\max}]$ and $n_{sd}^{\min} = 50, n_{sd}^{\max} = 80$ is given. The terminal condition is set to either the total elapsed time of 300 seconds or 8000 times of total iterated local search procedures applied, whichever comes earlier. The number of maximum generations to find rule gains $r_{\max}$ is set by either $r_{\max} = 0$ or $r_{\max} = 1$. Conventional local search method is represented by $r_{\max} = 0$ where no improvement with rule-based reconstruction is performed and the rule reconstruction procedure is added by setting $r_{\max} = 1$. When time window is removed, $c = 0.5$ and $r = 1.0$ is given in the priority rule-based reconstruction procedure instead of optimizing them by genetic algorithm. Binary tournament selection and uniform crossover are used in the genetic algorithm to find a set of time window and rule gains. Muta-

Table 3.1: Comparison results of performance of the priority rule-based reconstruction added to iterated local search (LA01-LA20)

Instance	n	m	$f = 1.3$			$f = 1.5$			$f = 1.6$		
			PRTW-ILS	PR-ILS	ILS	PRTW-ILS	PR-ILS	ILS	PRTW-ILS	PR-ILS	ILS
LA01	10	5	2299	2299	2299	1610	1610	1610	1230	1230	1230
LA02	10	5	1762	1762	1762	1028	1028	1028	695	695	695
LA03	10	5	1951	1951	1951	1280	1280	1280	1024	1024	1024
LA04	10	5	1917	1917	1917	1277	1277	1277	1029	1029	1029
LA05	10	5	1878	1878	1878	1205	1205	1205	877	877	877
LA06	15	5	5810	5813.5	5812.3	4593	4592.5	4594.1	4089.9	4093.4	4090
LA07	15	5	5798.3	5796.9	5811.2	4472.4	4473.2	4476.8	3863.2	3866.6	3868.8
LA08	15	5	5475	5475	5475	4034	4034	4034	3400	3400	3400
LA09	15	5	5608	5608	5608	4419	4419	4419	3811	3811.7	3811
LA10	15	5	6618	6618	6618	5148	5148	5148	4503	4503	4503
LA11	20	5	12143.3	12143.7	12124	10091.9	10167.3	10151	9244.4	9314.9	9279.3
LA12	20	5	10664.5	10664.6	10663.1	9089.5	9077.5	9054	8234.4	8260.2	8227.4
LA13	20	5	11577.3	11626.3	11601.3	9609.3	9653.9	9631.8	8754.6	8769.5	8759.8
LA14	20	5	13206.9	13210.1	13199.9	11449.8	11414.7	11426.2	10590.3	10573.2	10605.1
LA15	20	5	12449.3	12440.4	12474.4	10416.3	10381.1	10431.8	9451.7	9364.9	9412.4
LA16	10	10	1169	1169	1169	166	166	166	0	0	0
LA17	10	10	899	899	899	260	260	260	65	65	65
LA18	10	10	929	929	929.7	34	34	34	0	0	0
LA19	10	10	948	948	948	21	21	21	0	0	0
LA20	10	10	805	805	805	0	0	0	0	0	0

Table 3.2: Comparison results of performance of the priority rule-based reconstruction added to iterated local search (LA21-LA40)

Instance	n	m	$f = 1.3$			$f = 1.5$			$f = 1.6$		
			PRTW-ILS	PR-ILS	ILS	PRTW-ILS	PR-ILS	ILS	PRTW-ILS	PR-ILS	ILS
LA21	15	10	3598	3620.3	3638.3	1668.3	1660.5	1671.6	919.5	924.2	916.3
LA22	15	10	4368	4359.7	4377.1	2130.8	2100	2134.9	1317.2	1352	1386.8
LA23	15	10	3803.3	3861.1	3848	1510.6	1539.6	1510.3	932.1	935.4	935.7
LA24	15	10	3663.8	3698.9	3718	1571.5	1572.3	1578.3	756.5	756.9	753.6
LA25	15	10	3431.7	3395.5	3385.5	1430.5	1430	1430.1	869.2	875.8	877.8
LA26	20	10	9005.1	9157.9	9078.5	5930.5	5873.4	5846	4528.2	4563.6	4612.7
LA27	20	10	9374.8	9422.1	9465.8	5906.1	5907.4	5908.3	4342.4	4470.1	4460.9
LA28	20	10	9221.9	9475.5	9383.5	5667.3	5791.7	5925.7	4216.5	4245.3	4318.9
LA29	20	10	9124.8	9109.7	9089.9	5806.5	5846.6	5823.5	4495.1	4474.7	4537.3
LA30	20	10	8737.7	8813.7	8749.6	5285.7	5295.2	5258.9	3842.3	3836.3	3833.1
LA31	30	10	28369.1	29032.2	29315.7	22512.8	23677.3	23390.2	19955.9	20665.2	20992.3
LA32	30	10	30653.6	31903.9	31696.9	24366.1	25503	25970.1	21084	22489.3	22363.8
LA33	30	10	25837.6	27850.8	28227	20214.2	21540.1	22155.5	17599.2	19603.7	18942.3
LA34	30	10	27558.6	29294.8	28812.4	21814.4	22752.4	23745.3	19413.2	20471.7	21032.4
LA35	30	10	29671.8	30724.2	30785.8	23870.7	24230.4	24466.6	20682.2	21399.4	21966.9
LA36	15	15	3036.3	3028.2	3047.2	661.3	685	609.4	120.4	125.4	120.4
LA37	15	15	2471	2437.1	2436	414.4	414	412	0	0	0
LA38	15	15	2194.9	2255.2	2216.6	381.5	379.3	386.7	0	0	0
LA39	15	15	1709	1750.2	1730.9	0	0	0	0	0	0
LA40	15	15	2167.6	2181.4	2179.7	70.1	73.2	63.8	0	0	0

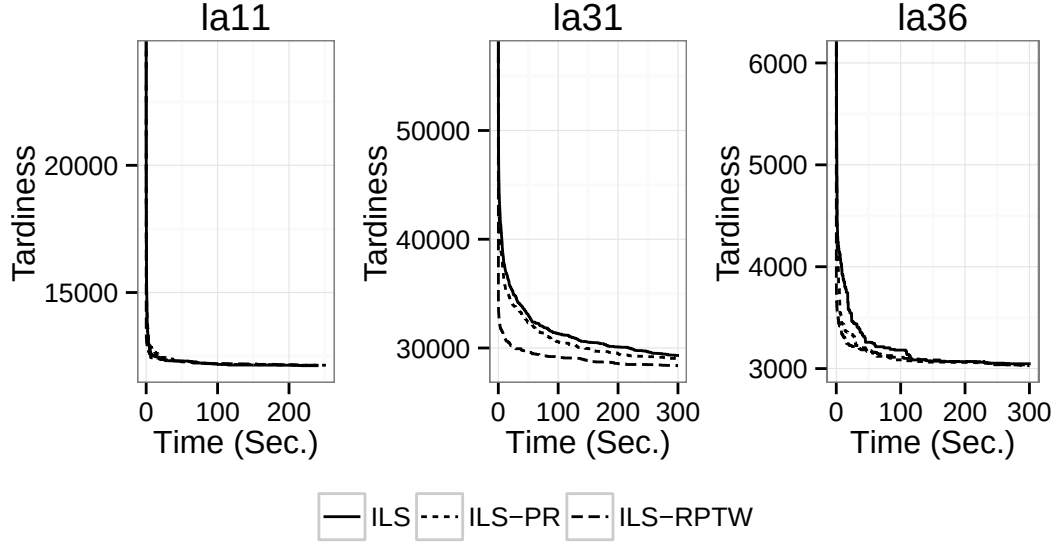


Figure 3.2: Comparison of decreasing speed of tardiness for benchmark instances with $f = 1.3$

tions are performed by updating one element of a chromosome by a uniform random number. Probability of crossover and mutation is set to 0.8 and 0.1 respectively. The proposed method is performed on benchmark instances 10 times independently with each of the parameter to collect the total weighted tardiness of a obtained solution.

Table 3.1 and Table 3.2 shows the results. RPTW-ILS denotes the average tardiness of solutions obtained by the iterated local search supplemented with priority-rule based reconstruction. PR-ILS shows the results of the iterated local search with priority-rule based reconstruction where time window is ignored and the whole sequence is reconstructed by priority rules. Finally, ILS shows the solution quality obtained by performing the existing iterated local search procedure multiple times iteratively. On 78 out of 120 problems, these methods obtained solutions with different average tardiness. RPTW-ILS acquired best average solutions on 47 problems, showing the efficiency of the priority rule-based reconstruction. RP-ILS and ILS obtained the best result on 15 and 19 problems, respectively. This shows that priority rule-based reconstruction procedure improves overall performance of the iterative local search procedure and that intense search realized by solution reconstruction

with time window is essential to achieve this improvement. On small instances shown in Table 3.1, the advantage of the PRTW-ILS over other methods is relatively weak. PRTW-ILS is particularly efficient on large scale problems in Table 3.2 and especially on problems where numbers of machines are greater than that of jobs. To illustrate convergence performance of each method, tardiness of the best solution and elapsed time are recorded each time iterated local search procedure is performed and the average values of ten trials are plotted in Figure 3.2. This figure shows that priority rule-based reconstruction tends to obtain better solution in the short term on large problems and finally obtained solutions is also better for the problems with larger number of machines compared to that of jobs.

3.3.2 Comparison with existing methods

The proposed PRTW-ILS is compared with GLS [EMDP08] which is one of the best existing methods on relatively small benchmark instances proposed by [PS99]. The maximum computation time is set to 10 seconds while the GLS is performed for 18 seconds on a personal computer with 2.8GHz processor. The result is shown in Table 3.3. For each instance, column Opt. shows results from the branch and bound method by [SP98] and column GLS shows the best and mean values on 10 independent runs reported by [EMDP08]. The proposed method is also performed on each instance for 10 times independently. The parameter which is confirmed to be efficient in the former experiment is used for the proposed method. The number of chromosomes is given by $g = 50$ and the number of maximum generation to find rule gains are fixed at $r_{\max} = 1$ from the former experiment. The maximum iteration number of conventional local search procedure n_{sd} is also given by a uniform random number $[n_{\text{sd}}^{\min}, n_{\text{sd}}^{\max}]$ where $n_{\text{sd}}^{\min} = 50, n_{\text{sd}}^{\max} = 80$ as suggested by [EMDP08]. Mean values of total weighted tardiness of the proposed method for each instance is estimated and it is tested whether there is significant difference from the reported mean value of GLS since only mean and best values of GLS are available. The proposed method significantly outperforms GLS on

Table 3.3: Comparison results of the proposed method with GLS on 10×10 benchmark instances

Instance	$f = 1.3$					$f = 1.5$					$f = 1.6$				
	opt.	PRTW-ILS		GLS		opt.	PRTW-ILS		GLS		opt.	PRTW-ILS		GLS	
		Mean	Best	Mean	Best		Mean	Best	Mean	Best		Mean	Best	Mean	Best
ABZ5	1405	1406	1403	1403	1403	69	69	69	69	69	0	0	0	0	0
ABZ6	436	436	436	436	436	0	0	0	0	0	0	0	0	0	0
LA16	1170	1169	1169	1175	1169	166	166	166	166	166	0	0	0	0	0
LA17	900	899	899	899	899	260	260	260	260	260	65	65	65	65	65
LA18	929	941.1	929	933	929	34	34	34	34	34	0	0	0	0	0
LA19	948	955.6	948	949	948	21	21	21	21	21	0	0	0	0	0
LA20	809	805	805	805	805	0	0	0	0	0	0	0	0	0	0
LA21	464	463	463	463	463	0	0	0	0	0	0	0	0	0	0
LA22	1068	1077.4	1064	1087	1064	196	196	196	196	196	0	0	0	0	0
LA23	837	846.4	835	865	835	2	2	2	2	2	0	0	0	0	0
LA24	835	835	835	835	835	82	84.4	82	86	82	0	0	0	0	0
MT10	1368	1363	1363	1372	1363	394	394	394	394	394	141	144.4	141	162	141
ORB01	2568	2638.5	2568	2651	2570	1098	1208.3	1130	1159	1098	566	633.4	566	688	604
ORB02	1412	1420.5	1408	1444	1408	292	292	292	292	292	44	44	44	44	44
ORB03	2113	2145.3	2111	2170	2111	918	947.9	918	943	918	422	454.8	422	514	422
ORB04	1623	1658.2	1623	1643	1623	358	358	358	394	358	66	66	66	78	66
ORB05	1593	1714.2	1593	1659	1593	405	412.7	405	405	405	163	175.4	163	181	181
ORB06	1792	1790	1790	1790	1790	426	426	426	440	426	31	28	28	28	28
ORB07	590	590	590	592	590	50	50	50	55	50	0	0	0	0	0
ORB08	2429	2482.7	2429	2522	2439	1023	1026.6	1023	1059	1023	612	633.5	621	669	646
ORB09	1316	1316	1316	1316	1316	297	315	297	311	297	66	86.1	66	83	66
ORB10	1679	1738.4	1679	1718	1679	346	423.4	372	400	346	76	98.1	76	142	84

12 out of 66 instances while GLS significantly performs better on 2 instances with significance level of 5%. The result with significant difference are written in bold. Therefore, the performance of the proposed method is competitive with GLS on the benchmark set.

To evaluate performance of the proposed method with sufficient computational time, medium sized benchmark instances proposed by [EMDP08] is tested. This benchmark set includes more difficult instances with up to 30 jobs and 10 machines. LA16-LA20 are overlapped in the previous benchmark. Since GLS is performed on these instances for 200 generations with population number of 40, iterated local search procedures are performed on each chromosome on each generation for the total of 8,000 times. Therefore, the terminal condition of the proposed method is also set to the 8,000 times of the iterated local search procedure performed. This makes the computational efforts for the both methods roughly the same because computational time of the both methods are generally dominated by the time for the local search procedure. In addition to that, to perform sufficient trials on each problem, the proposed method is also terminated if 300 seconds of the elapsed time is reached even when the terminal condition by the numbers of local search procedures applied is not met. The parameters of the proposed method are the same as the previous experiment and both methods are performed on each instance 10 times. The results are presented in Table 3.4 and Table 3.5. Please note that the experiments conditions for the proposed method is the same as that for the previous experiment to compare the efficiency of the rule-based reconstruction procedure. Therefore, the results shown in these table as columns PRTW-ILS are identical to that of the previous experiment except that their significance is compared with the results of GLS. Both methods yield optimal solution for LA16-LA20 in every trial. This implies that computational time is not sufficient for the both methods in the previous experiment. On small instances with $n \leq 10$ and $m \leq 10$ including LA16-LA20, both methods are equally efficient. On large instances, however, the proposed method is considerably more efficient exceeding the best solution found by GLS on

Table 3.4: Comparison results of the proposed method with GLS on medium sized benchmark instances (LA01 - LA20)

Instance	n	m	$f = 1.3$				$f = 1.5$				$f = 1.6$			
			PRTW-ILS		GLS		PRTW-ILS		GLS		PRTW-ILS		GLS	
			Mean	Best	Mean	Best	Mean	Best	Mean	Best	Mean	Best	Mean	Best
LA01	10	5	2299	2299	2299	2299	1610	1610	1610	1610	1230	1230	1230	1230
LA02	10	5	1762	1762	1762	1762	1028	1028	1028	1028	695	695	695	695
LA03	10	5	1951	1951	1951	1951	1280	1280	1280	1280	1024	1024	1024	1024
LA04	10	5	1917	1917	1917	1917	1277	1277	1277	1277	1029	1029	1029	1029
LA05	10	5	1878	1878	1878	1878	1205	1205	1205	1205	877	877	877	877
LA06	15	5	5810	5810	5827	5810	4593	4592	4658	4592	4089.9	4089	4130	4098
LA07	15	5	5798.3	5765	5801	5765	4472.4	4470	4548	4470	3863.2	3851	3988	3843
LA08	15	5	5475	5475	5482	5475	4034	4034	4094	4034	3400	3400	3400	3400
LA09	15	5	5608	5608	5648	5608	4419	4419	4421	4419	3811	3811	3835	3811
LA10	15	5	6618	6618	6621	6618	5148	5148	5148	5148	4503	4503	4533	4503
LA11	20	5	12143.3	12024	12341	12039	10091.9	10054	10332	10170	9244.4	9232	9399	9232
LA12	20	5	10664.5	10608	10683	10542	9089.5	9011	9084	8982	8234.4	8203	8302	8255
LA13	20	5	11577.3	11502	11889	11557	9609.3	9594	9864	9626	8754.6	8680	8916	8767
LA14	20	5	13206.9	13128	13225	13107	11449.8	11366	11382	11286	10590.3	10491	10594	10496
LA15	20	5	12449.3	12380	12428	12330	10416.3	10157	10455	10343	9451.7	9320	9392	9284
LA16	10	10	1169	1169	1169	1169	166	166	166	166	0	0	0	0
LA17	10	10	899	899	899	899	260	260	260	260	65	65	65	65
LA18	10	10	929	929	929	929	34	34	34	34	0	0	0	0
LA19	10	10	948	948	948	948	21	21	21	21	0	0	0	0
LA20	10	10	805	805	805	805	0	0	0	0	0	0	0	0

Table 3.5: Comparison results of the proposed method with GLS on medium sized benchmark instances (LA21 - LA40)

Instance	n	m	$f = 1.3$				$f = 1.5$				$f = 1.6$			
			PRTW-ILS		GLS		PRTW-ILS		GLS		PRTW-ILS		GLS	
			Mean	Best	Mean	Best	Mean	Best	Mean	Best	Mean	Best	Mean	Best
LA21	15	10	3598	3598	3771	3560	1668.3	1636	1692	1570	919.5	896	949	868
LA22	15	10	4368	4282	4471	4227	2130.8	2027	2273	2015	1317.2	1242	1450	1242
LA23	15	10	3803.3	3777	3955	3777	1510.6	1510	1683	1510	932.1	930	977	936
LA24	15	10	3663.8	3543	3831	3553	1571.5	1570	1618	1570	756.5	750	773	752
LA25	15	10	3431.7	3313	3569	3313	1430.5	1430	1497	1430	869.2	828	922	874
LA26	20	10	9005.1	8847	9748	9093	5930.5	5825	6106	5369	4528.2	4267	5125	4159
LA27	20	10	9374.8	9231	9860	9127	5906.1	5663	6142	5793	4342.4	4223	4590	4307
LA28	20	10	9221.9	8985	9757	9263	5667.3	5442	6254	5744	4216.5	3936	4594	4248
LA29	20	10	9124.8	8988	9397	8744	5806.5	5680	6392	5959	4495.1	4406	4706	4392
LA30	20	10	8737.7	8636	8968	8626	5285.7	5090	5496	5259	3842.3	3685	4131	3631
LA31	30	10	28369.1	27370	28999	27671	22512.8	21696	23417	22545	19955.9	19203	21141	20006
LA32	30	10	30653.6	30074	32004	30301	24366.1	23810	25766	24467	21084	20297	22918	21722
LA33	30	10	25837.6	25461	27216	25902	20214.2	19368	21767	20493	17599.2	17127	18801	17771
LA34	30	10	27558.6	27089	29131	27901	21814.4	21147	23341	22572	19413.2	19057	20548	19693
LA35	30	10	29671.8	28850	30979	29652	23870.7	23323	24654	24325	20682.2	20457	21991	20601
LA36	15	15	3036.3	2998	3187	2957	661.3	590	866	590	120.4	118	192	116
LA37	15	15	2471	2361	2880	2290	414.4	413	589	413	0	0	0	0
LA38	15	15	2194.9	2109	2287	2159	381.5	316	438	401	0	0	0	0
LA39	15	15	1709	1675	1861	1676	0	0	9	0	0	0	0	0
LA40	15	15	2167.6	2143	2305	2078	70.1	57	79	51	0	0	0	0

38 instances. Estimated mean values of total weighted tardiness acquired by the proposed method are compared to reported mean values of GLS as with smaller benchmark problems. The result shows that the proposed method outperforms GLS in 67 instances and that GLS performs better in 2 instances with significance level of 5%. Significantly different results are shown in bold in Table 3.4 and Table 3.5. This is because the selection procedure in GLS does not consider the diversity of solutions and thus premature convergence can occur. On the other hand, the proposed method applies the rule-based reconstruction procedure which can impose a substantial change to the temporal solution and enable to search diverse solution by local search procedure.

3.4 Conclusion

Priority rule-based reconstruction method integrated with the conventional local search method for JSP with total weighted tardiness criterion is proposed. The proposed method was compared with state-of-the-art approach for JSP with tardiness criterion and the result shows that the proposed method is superior given that sufficient computational time is allowed. The rule-based reconstruction procedure modifies the operation sequences within an arbitrary selected time window and this enables to deal with the solution representation typically used in local search methods. In the reconstruction procedure, linear combination of simple priority rules is adopted. The gains are optimized by genetic algorithm along with the time window to get a better solution and the number of chromosomes defines the computational effort dedicated to the reconstruction procedure. This reconstruction procedure and conventional local search procedure are alternately applied to the temporal solution. Experiment results showed the rule-based reconstruction procedure improves the overall performance. Finally, our proposal method was applied to the benchmark instances proposed by [EMDP08] and the proposal method yielded considerably better solutions on these instances than their efficient GLS method.

To improve our method further, several perspectives can be considered.

First, components of the linear combined priority rule are arbitrarily selected from relatively simple priority rules and more sophisticated rules from recent studies can improve the performance of the rule-based reconstruction procedure. Optimization of the gains of linear combined rule by genetic algorithm becomes difficult with the increasing number of its component. Therefore, components of the linear combined rule should be carefully selected. Additionally, the proposed method can be used for other criteria with appropriate priority rules. For example, makespan minimization can be considered by exchanging due date related priority rules with priority rules which regards total remaining operation time or number of remaining operations. Finally, the amount of computational effort dedicated to rule-based reconstruction procedure needed for maximum overall performance is still unknown. In the situation that the rule-based reconstruction is likely to improve the quality of the temporal solution, extensive search of the time window and rule gains should be performed and iteration number of local search should be increased otherwise.

Bibliography

- [EMDP08] Imen Essafi, Yazid Mati, and Stéphane Dauzère-Pérès. A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. *Comput. Oper. Res.*, 35(8):2599–2616, August 2008.
- [GT60] B. Giffler and G. L. Thompson. Algorithms for solving production-scheduling problems. *Operations Research*, 8(4):pp. 487–503, 1960.
- [Kre00] Stephan Kreipl. A large step random walk for minimizing total weighted tardiness in a job shop. *Journal of Scheduling*, 3(3):125–138, 2000.
- [PI77] S. S. Panwalkar and Wafik Iskander. A survey of scheduling rules. *Operations Research*, 25(1):45–61, 1977.

- [PS99] Michael Pinedo and Marcos Singer. A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop. *Naval Research Logistics (NRL)*, 46(1):1–17, 1999.
- [SP98] Marcos Singer and Michael Pinedo. A computational study of branch and bound techniques for minimizing the total weighted tardiness in job shops. *IIE Transactions*, 30(2):109–118, 1998.
- [TYSF13] Yasumasa Tamura, Masahito Yamamoto, Ikuo Suzuki, and Masashi Furukawa. Acquisition of dispatching rules for job-shop scheduling problem by artificial neural networks using pso. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 17(5):731 – 738, 2013.
- [VM87] Ari P. J. Vepsalainen and Thomas E. Morton. Priority rules for job shops with weighted tardiness costs. *Management Science*, 33(8):pp. 1035–1047, 1987.
- [ZCL09] Hong Zhou, Waiman Cheung, and Lawrence C. Leung. Minimizing weighted tardiness of job-shop scheduling using a hybrid genetic algorithm. *European Journal of Operational Research*, 194(3):637 – 649, 2009.
- [ZSW13] Rui Zhang, Shiji Song, and Cheng Wu. A dispatching rule-based hybrid genetic algorithm focusing on non-delay schedules for the job shop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, 67(1-4):5–17, 2013.

Chapter 4

Priority rule-based method for Flexible Job-shop Scheduling Problem

4.1 Introduction

When the real production environment is optimized, constraints of schedules are generally much more complex than the classical JSP, so flexible JSP (FJSP) which is an extension of the classical JSP has also been studied. Each operation in an FJSP is processed on given multiple machines rather than on a single predetermined machine as is done in the classical JSP and one of these machines must be selected to obtain a feasible schedule. The processing time of the same operation may also differ with the machine to be processed, so heuristic methods used for the classical JSP can be applied to FJSPs with modifications of the machine assignment. Neighborhood structures of local search methods used for the FJSPs consider additional machine assignment changes [MG00] and genetic algorithms for the FJSP use an additional vector of machine assignments to encode a solution into a chromosome [PMC08, ZGS11]. Thus far, hybrid methods of evolutionary computation and local search are considered to be among the most effective methods for FJSPs [GSG08, YX13].

The new heuristic method presented for FJSPs involves a priority rule-based construction (PRC) procedure in which a linearly weighted sum of priority rules for the FJSP is proposed and integrated into a hybrid genetic algorithm (HGA) with a local search. The importance of each priority rule is denoted by a weight, and different sets of weights construct different solutions, so weights are optimized along with populations of schedules. For most existing algorithms for FJSPs, priority rules are used for creating the initial population [PMC08, ZGS11]. In such initializing methods, Learnable Genetic Architecture (LEGA) by Ho et al. [HTL07] is remarkable in that it selects processing machine of operations simultaneously determining processing sequences of operations on each machine. Tay and Ho also proposed a composite priority rule and its optimization by genetic programming [TH08]. The proposed construction procedure determines machine assignment and processing sequence simultaneously similar to LEGA and uses a combination of multiple basic priority rules as well as the genetic programming approach. The genetic programming approach is different from the proposed method, however, in that it solves the multi-objective FJSP rather than the minimization of the makespan and in that the proposed method uses a linearly weighted sum instead of a tree-based representation to decrease computational time. In addition to that, the proposed method is integrated with meta-heuristic approaches which consists of local search and genetic algorithm to improve its efficiency.

4.2 Proposed Approximation Algorithm

PRC procedure is introduced to complement existing HGA and to further improve the solution. The priority rule is represented by a linearly weighted sum of basic rules, and its weights are optimized by using a genetic algorithm to construct a better solution separately from solution optimization. An individual solution in a population of HGAs is also replaced if the priority rule constructs a better schedule so that the resulting solution can be further

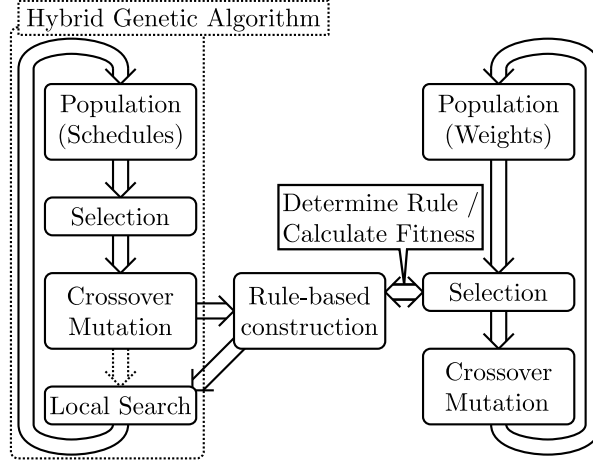


Figure 4.1: Overview of the proposed method

improved through genetic operators or local search procedures.

The proposed method consists of the three components shown in Fig. 4.1. Its main component is the PRC procedure in the center of the diagram, which uses a schedule and a set of weights representing a certain priority rule to construct another schedule. Schedules are optimized through the conventional HGA shown at left in the diagram. Schedules can also be improved by using the rule-based construction procedure once in a generation. If a schedule constructed by a construction procedure does not improve the makespan of the original schedule, the original schedule is left unchanged as indicated by the dotted arrow. The weights in the construction procedure are optimized by using another genetic algorithm loop, shown right in the diagram.

4.2.1 PRC

The proposed construction procedure determines the processing sequence of operations on each machine by a linearly weighted sum of basic priority rules. For higher machine utilization, each operation is considered on all machines where the operation is processed by problem constraints. Because operations processed on a small number of machines have less chance of preceding other operations by the very fact that they are considered on fewer machines, a *primary machine* is selected for each operation and operations on primary

machines are processed earlier than other operations.

Primary machine $M'(o_{ij})$ for each operation o_{ij} is determined by calculating priority value r_{ijk} on each machine $M_k \in A_{ij}$ and selecting the machine with the highest priority value,

$$r_{ijk} = \alpha_1 \frac{p_{ijk}}{\max_{M_p \in A_{ij}} p_{ijp}} + \alpha_2 \frac{W_k}{\max_{M_p \in M} W_p}$$

where workload W_k is the total processing time of operations processed on M_k . The workload is imposed to determine machine assignment according to the previous schedule and to increase machine utilization. Note that this procedure requires a feasible schedule as input to calculate the machine workload. In the proposed method, the feasible schedule is given by an individual solution in a population of HGAs.

The solution is then reconstructed by deciding the processing sequence on each machine as follows:

1. Initialize sets of assigned operations $F = \emptyset$ and assignable operations $U = \{o_{11}, o_{21}, \dots, o_{n1}\}$.
2. Consider assigning each assignable operation $o_{ij} \in U$ on each assignable machine $M_k \in A_{ij}$ as early as possible and letting the earliest starting time be s'_{ijk} , then obtain earliest assignment $o_{pq} \in U$ and M_r whose completion time $s'_{pqr} + p_{pqr}$ is the minimum.
3. Acquire set of conflicting operations C whose earliest possible assignment overlaps with o_{pq} on M_r .
4. If operation $o_{pq} \in C$ whose primary machine assignment $M'(o_{pq}) = M_r$ exists, all operations not satisfying this condition are removed from C .
5. Select an operation from C based on the priority rules below.
6. Assign the selected operation and update assigned set F . Add a subsequent operation of the selected operation to U if it exists.
7. Repeat the above steps until all operations are assigned.

To resolve conflicts in step 4, the weighted sum of priority rules with adaptive weights proposed by Tamura et al. [TYSF13] is used.

$$u_{ijk} = \alpha_3 \frac{p_{ijk}}{\max_{s_{pq} \in C} p_{pqr}} + \alpha_4 \frac{R_{ij}}{\max_{s_{pq} \in C} R_{pq}} + \alpha_5 \frac{I_{ijk}}{\max_{s_{pq} \in C} I_{pqr}}$$

where R_{ij} is estimated remaining time $\sum_{p=j+1}^{n_i} p_{ipM'(o_{ip})}$ before job J_i is completed and I_{ij} is the machine's estimated idle time. The parameters are described in [TYSF13].

The obtained solution is determined by weights $\alpha_1, \alpha_2, \dots, \alpha_5$. To obtain a solution with a better quality, these values are optimized together with the schedule in the proposed method.

4.2.2 Integration with a HGA

Because the rule-based construction described above is intended to complement existing local search methods, the procedure is implemented to enhance the HGA, which is known to be one of the most effective approaches to solving the FJSP. Using the HGA improves each chromosome until local optima are found by a local search in every generation. The rule-based construction procedure is also applied to each chromosome in the proposed method and replaces a chromosome if the obtained schedule has a shorter makespan. Weights are also optimized by a GA together with solution chromosomes to find a better solution with the procedure.

A parameter set for the construction procedure is represented by a vector of five real numbers $(\alpha_1, \alpha_2, \dots, \alpha_5)$ within a range of $[-1.0, 1.0]$ for each parameter. When the procedure is applied to a solution, weights are directly set by α_i . The number of parameter chromosomes is the same as solution chromosomes and each parameter chromosome corresponds to one randomly selected solution chromosome. The fitness value of the parameter chromosome is given by the solution's makespan after the construction procedure is applied with the corresponding parameter set.

Encoding schemes of solution chromosomes and local search methods are the same to the method described in Chapter 2. The overall framework of the proposed method is as follows:

1. Initialize solution chromosomes and parameter chromosomes.
2. Decode each solution chromosome and generate a new solution by applying the construction procedure with parameters from the corresponding parameter chromosome.
3. Set the fitness value of the parameter chromosome by using makespan of the generated solution.
4. Update the solution chromosome if the generated solution has a makespan smaller than that of the previous solution.
5. Apply the local search method to each solution chromosome and set its fitness value by using the makespan of the resulting schedule.
6. Apply genetic operations (selection, mutation, and crossover) to both chromosomes independently.
7. Return to step 2 and iterate generation until terminal conditions are satisfied.

The method proposed above and the HGA without the construction procedure is implemented to compare the performance of the reconstruction procedure. In the method without construction, parameter chromosomes and steps 3, 4, and 5 is omitted.

4.3 Performance Evaluation

To assess the performance of the rule-based construction, Both the proposed method and the hybrid genetic method without the construction procedure is applied to benchmark problems and tested whether the average makespan of the solutions obtained by the proposed method is significantly better than the

solutions acquired by the hybrid genetic method. The results on the benchmark problems implies that the proposed method performs better than the hybrid genetic method on some problems with particular conditions satisfied. Therefore, the proposed method and the hybrid genetic method are also performed on randomly generated problems with different properties in order to investigate conditions where the proposed method have an advantage over the existing method.

4.3.1 Experiments on benchmark problems

Experiment conditions

The proposed method with PRC is applied to each benchmark problem. The proposed method without the construction procedure and parameter chromosomes (HGA) is also implemented and the makespan of the obtained solution is compared to that of PRC. The parameters for genetic operations shared between them are as follows:

- Number of chromosomes : 50
- Maximum generation : 200
- Crossover probability : 0.7
- Mutation probability : 0.1

If the probabilities of applying genetic operators are too low, it is difficult to escape from local optima of local search procedures and the quality of the obtained solutions tends to be low. If the probabilities are too high, local searches cannot concentrate on promising regions of the solution space, which leads to longer computation time for local search procedures because larger numbers of iterations are needed to find a local optimum. This means that the probabilities are empirically determined to acquire good solutions within a reasonable time. Parameters for optimizing the weights of the linearly weighted rule in PRC are set as listed below. These parameters should be adjusted to obtain an effective priority rule that improves the solutions of the

HGA, so these parameters are also determined empirically to maximize PRC efficiency.

- Parameter crossover probability : 0.7
- Parameter mutation probability : 0.1

Experiments are performed on three benchmark sets commonly used to evaluate the performance of algorithms for the FJSP: BRdata, DPdata and HUdata.

1. HUdata : These benchmark problems are derived from benchmark problems for the classical JSP [HJT94]. Machines assignable for each operation are duplicated and their processing time is identical to that of the original machine. Three subsets – edata, rdata and vdata – are included and the average numbers of assignable machines for operations $|A_{ave}|$ are 1.15, 2 and $\frac{1}{2}m$. m is the number of machines.
2. BRdata : This benchmark set includes 10 randomly generated problems [Bra93]. The maximum number of machines that can process an operation is given and processing times on different machines are generated randomly with uniform distribution.
3. DPdata : This benchmark set includes 18 randomly generated problems [DPP97]. The differences from BR data are that the processing time of an operation on different machines are either equivalent to or has a small deviation compared to the overall deviation in processing time.

Both methods are implemented in C++ and following numerical experiments are performed on a 3.3 GHz Intel Core i5 processor and 16GB of RAM. Both algorithms use random numbers in their procedures, so their performance must be discussed in a statistical test among the multiple solutions obtained by each method. To do so, each method is performed on every problem 100 times and then we applied Student's t-test to the makespan of solutions obtained by each method to test whether their mean value differs at a certain level of significance.

Table 4.1: Number of problems where PRC performed better / worse than HGA.

Benchmark		Number	Better	Worse
HUdata	edata	43	13	7
	rdata	43	0	17
	vdata	43	14	10
BRdata		10	3	2
DPdata		18	4	8

Result of the experiment

The average makespan of obtained PRC and HGA solutions is compared and it is tested whether these values differ significantly among these methods at a significance level of 1%. Table 4.1 shows the number of problems for each benchmark set and for when where PRC is significantly better or worse than HGA. No significant difference was seen in other cases.

Results for HUdata show that PRC has a negative impact on the quality of solutions to problems with a small number of average assignable machines $|A_{ave}|$. For edata with $|A_{ave}| = 1.15$, PRC has a positive effect on the average quality and PRC fails to obtain good result on with larger $|A_{ave}|$. Regarding problems for HUdata, machine assignments are determined by machine workload alone because the processing time on all machines is equivalent. This implies that PRC cannot determine the effective machine assignment for HUdata because machine assignments in the proposed method consider only machine workloads and processing time, one of which is essentially constant for HUdata.

For BRdata, PRC fails to improve overall quality despite the fact that they have different processing time and thus machine assignments are determined by both workload and processing time. One possible explanation for this is that the deviation in processing time on different machines is so large that assignments on machines with large processing times are often impractical to shorten makespans and machine assignments should be determined solely by processing time.

Results for DPdata may be most interesting because PRC acquired better solutions than state-of-the-art HGA for a few problems. Details of results

Table 4.2: Performance comparison for DP data

Problem	Parameters			HGA			PRC			R_{eff}	Gao et. al.	
	$m \times n$	Δp_i	P	Best	Ave.	Time	Best	Ave.	Time		Best	Ave.
01a	10×5	0	0.1	2523	2563.05	3.23	2526	2570.50	3.90	-0.29%	2518	2518.0
02a	10×5	0	0.3	2230	2243.46	6.09	2232	2246.56	7.09	-0.14%	2231	2531.0
03a	10×5	0	0.5	2229	2233.01	8.80	2229	2235.59	9.31	-0.12%	2229	2229.3
04a	10×5	5	0.1	2510	2550.96	3.23	2510	2557.78	3.88	-0.27%	2515	2518.0
05a	10×5	5	0.3	2218	2231.92	6.13	2216	2232.20	6.77	-	2217	2218.0
06a	10×5	5	0.5	2198	2210.01	9.19	2185	2205.93	9.88	0.18%	2196	2198.0
07a	15×8	0	0.1	2328	2407.92	6.00	2349	2404.07	7.33	-	2307	2309.8
08a	15×8	0	0.3	2067	2079.96	17.84	2071	2087.11	19.97	-0.34%	2073	2076.0
09a	15×8	0	0.5	2064	2070.67	24.03	2064	2070.25	18.97	-	2066	2067.0
10a	15×8	5	0.1	2322	2397.45	6.03	2340	2399.96	7.32	-	2315	2315.2
11a	15×8	5	0.3	2061	2076.66	17.85	2061	2085.03	21.02	-0.40%	2071	2072.0
12a	15×8	5	0.5	2032	2044.07	25.05	2005	2024.43	19.58	0.96%	2030	2030.6
13a	20×10	0	0.1	2290	2327.95	11.65	2296	2330.36	13.57	-	2257	2260.0
14a	20×10	0	0.3	2167	2174.16	36.97	2170	2178.03	33.37	-0.18%	2167	2167.6
15a	20×10	0	0.5	2165	2168.82	46.10	2164	2166.83	30.76	0.09%	2165	2165.4
16a	20×10	5	0.1	2289	2320.11	11.45	2292	2322.93	13.28	-	2256	2258.0
17a	20×10	5	0.3	2143	2155.34	36.48	2135	2159.60	33.58	-0.20%	2140	2142.0
18a	20×10	5	0.5	2133	2144.86	46.04	2091	2104.66	32.11	1.87%	2127	2130.7

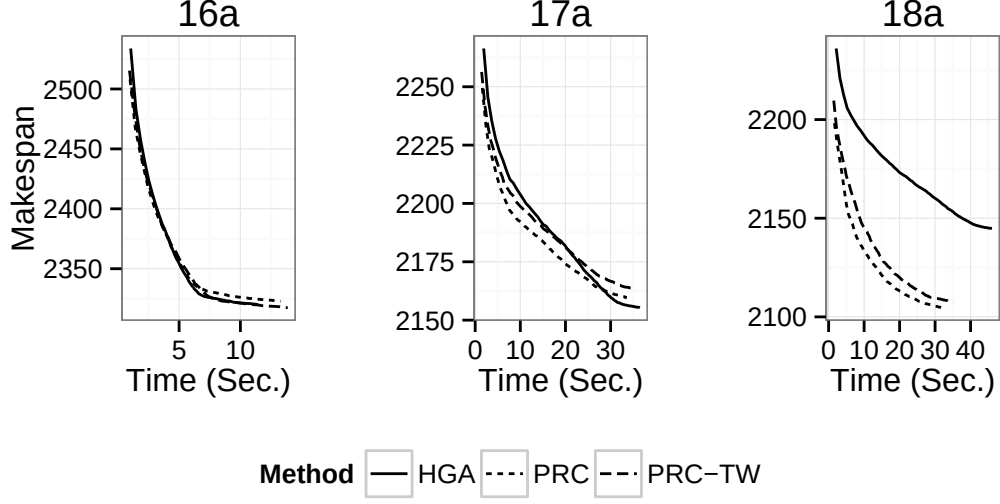


Figure 4.2: Comparison of decreasing speed of makespan for DP data

and the property of each problem are shown in Table 4.2. The number of operations for each job is determined randomly within the value of n_i . Each machine is set to be assignable for each operation with the probability of P . The maximum deviation in processing time from the average processing time of the operation is Δp_i , where the average processing time is determined randomly in a range of $[10, 100]$ for all problems. The quality of solutions by RPC and HGA is compared to verify the effectiveness of the proposed construction procedure. For comparing their efficiency to that of the state-of-the-art algorithm, results reported by Gao et al. [GSG08] that use extensive local search procedures are shown in Table 4.2. Note that the results by Gao et al. shows results for five trials whereas other columns show results for 100 trials, with the best results among the algorithms shown in bold.

The effectiveness rate of PRC R_{eff} is calculated for the problems for which results by PRC and HGA differs significantly. This effectiveness rate is calculated by $R_{\text{eff}} = 100 \times (1 - C_{\text{ave}}^{\text{PRC}} / C_{\text{ave}}^{\text{HGA}})$ where $C_{\text{ave}}^{\text{PRC}}$ and $C_{\text{ave}}^{\text{HGA}}$ is the average makespan of the obtained solution by using PRC and HGA, so higher R_{eff} means PRC performed better than HGA. PRC outperforms an algorithm by Gao et al. on two problems that have the largest number of assignable

operations and highest deviation in processing time. This is because machine assignments are selected appropriately by using the proposed priority rule due to a deviation in processing time, and optimal machine assignments are difficult to obtain because of the relatively small deviation. PRC has a computation time considerably shorter than HGA for the problems that PRC performs better than HGA. This is interesting because PRC performs additional construction procedure on top of the algorithm of HGA. In this situation, the construction procedure effectively decreases the makespan and solutions converge at local optima with fewer iterations when local search procedure is applied. Additionally, average makespan of solutions according to elapsed time is shown for each method in Figure 4.2 to clarify superiority of PRC over HGA. This figure shows that the rule-based reconstruction improves both convergence speed and the quality of the obtained solution on the problem mentioned above.

Overall, experiments on benchmark problems showed that PRC tended to be effective for some problems and outperformed the state-of-the-art algorithm for a particular problem. The problems for which PRC performed better the meet following conditions:

- The average number of assignable machines is large.
- Deviation in processing time on different machine on a operation is not zero and is smaller than the deviation in average processing time of operations.

4.3.2 Experiments of long-term performance on problems with various properties

Experiment conditions

The experiments on benchmark problems revealed that the proposed method is remarkably better than existing methods on some problems in DPdata which have some shared properties among them. PRC and HGA are performed with a different terminal condition that allows larger number of gen-

erations in order to investigate long-term performance of PRC. Therefore, the terminal condition is a specific elapsed time and other conditions are the same as the previous experiments.

To clarify the conditions under which PRC performs better than conventional searches, problems with various numbers of assignable machines and deviations in processing time on different machines are newly generated. In addition to the problems from DPdata, PRC and HGA are also applied to these generated problems and their results are compared. The number of jobs n , machines m and operation per job n_i is the same for all problem with $n = 20, m = 10$ and $n_i = [10, 15]$. The number of assignable machines for each operation is determined randomly with uniform distribution within $[1, 2], [1, 6]$ or $[1, 10]$. The standard processing time for each operation $t_{o_{i,j}}$ is determined randomly within $[10, 100 - \Delta p_i]$ where Δp_i is a maximum deviation of processing time among different machines. Then, processing time of the operation on each machine is randomly given by $t_{o_{i,j}} + t_m$ where t_m is a randomly generated value with uniform distribution $[0, \delta p_i]$.

The PRC applies the priority rule-based reconstruction to replace individuals in the hybrid genetic algorithm by determining machine assignments and processing sequences of all operations. This is because the FJSP has larger number of feasible solution than the JSP with similar size and it is expected that diversification is more important than intensification in order to search sub-optimal solutions for the FJSP. To confirm the expectation, PRC-TW where a part of operations in a solution is selected by time window and then machine assignments and processing sequence of the selected operations is also applied to the problems to represent intensifying search by the priority rule-based reconstruction. In PRC-TW, center of time window c and the width of the time window r that are represented by ratio to makespan are optimized along with the priority rule weights $\alpha_1, \alpha_2, \dots, \alpha_5$.

Both methods are implemented in C++, and following numerical experiments are performed on a 1.6 GHz Intel Xeon processor and 32GB of RAM. Both algorithms use random numbers in their procedures, so their perfor-

mance must be discussed in a statistical test among the multiple solutions obtained by each method. To do so, we applied Student's t-test to the makespan of solutions obtained by PRC and HGA to test whether their mean value differs at a certain level of significance.

Results of the experiment

Table 4.3 shows the results where PRC or HGA is applied to DPdata with the long-term terminal condition described above. R_{eff} is only shown for the problems for which results by PRC and HGA differ significantly with significance level of 5% and the column Gen. shows the average number of generation of the genetic algorithm when the terminal condition is met. Figure 4.3 shows the average makespan of solutions according to elapsed time on each generation. The results shows that the HGA quickly converges to local optima on specific problems where PRC is superior to the HGA.

The average makespans of obtained solutions on newly generated problems with various numbers of processing machine and deviation of processing time among machines are shown in Table 4.4. As expected, PRC performs well on problems with many assignable machines and little deviation. It is confirmed that PRC is less efficient for problems with higher deviation and zero deviation. PRC is the most efficient on problems whose deviation of processing time on different machines is about 30% of the deviation for all processing time. The average makespans of solutions according to elapsed time is shown in Figure 4.4. This figure shows that PRC improve convergence speed for a problem with zero deviation although the accuracy of the obtained solution is same when the algorithms are performed for a sufficient period of time. It is also revealed that intensifying search by PRC-TW fails to find better solutions than PRC in most case. In particular, PRC is remarkably better than PRC-TW on the problems where the priority rule-based reconstruction works well and the solutions are largely improved by PRC.

Based on these results, we concluded that the effectiveness of the proposed method depends on the number of possible machine assignments in semi op-

Table 4.3: Performance comparison for DP data in long-term

Problem	Parameters			HGA			PRC			R_{eff}	Gao et. al.		PRC-TW		
	$m \times n$	Δp_i	P	Best	Ave.	Gen.	Best	Ave.	Gen.		Best	Ave.	Best	Ave.	Gen.
01a	10×5	0	0.1	2505	2546.28	4451.12	2505	2552.88	4449.62	-0.26%	2518	2518.0	2507	2546.83	4477.75
02a	10×5	0	0.3	2231	2238.79	3461.89	2230	2239.50	3446.05	-	2231	2531.0	2230	2241.26	3494.25
03a	10×5	0	0.5	2228	2230.53	2531.51	2228	2233.30	2635.03	-0.12%	2229	2229.3	2228	2231.72	2501.01
04a	10×5	5	0.1	2503	2530.76	4607.62	2506	2543.53	4759.39	-0.50%	2515	2518.0	2503	2531.81	4541.93
05a	10×5	5	0.3	2213	2226.71	3487.86	2216	2227.05	3447.96	-	2217	2218.0	2215	2228.77	3526.68
06a	10×5	5	0.5	2195	2204.81	2633.49	2183	2198.04	2857.28	0.31%	2196	2198.0	2186	2204.86	2652.88
07a	15×8	0	0.1	2318	2373.15	2668.15	2334	2376.41	2565.03	-	2307	2309.8	2286	2370.69	2658.56
08a	15×8	0	0.3	2068	2076.75	1631.89	2068	2081.36	1591.25	-0.22%	2073	2076.0	2069	2079.09	1546.04
09a	15×8	0	0.5	2064	2068.90	1136.29	2063	2067.53	1305.23	0.07%	2066	2067.0	2063	2067.67	1262.60
10a	15×8	5	0.1	2311	2369.36	2656.93	2304	2372.95	2633.47	-	2315	2315.2	2300	2373.77	2692.86
11a	15×8	5	0.3	2060	2072.65	1614.66	2060	2077.35	1641.72	-0.23%	2071	2072.0	2065	2078.01	1616.82
12a	15×8	5	0.5	2029	2041.25	1195.33	2004	2015.12	1384.96	1.28%	2030	2030.6	2001	2020.06	1326.52
13a	20×10	0	0.1	2279	2311.91	1647.41	2287	2315.84	1629.68	-0.17%	2257	2260.0	2286	2312.58	1667.12
14a	20×10	0	0.3	2166	2173.44	686.94	2167	2174.99	776.31	-0.07%	2167	2167.6	2167	2176.74	710.91
15a	20×10	0	0.5	2164	2168.78	450.59	2163	2165.73	686.39	0.14%	2165	2165.4	2163	2166.01	672.18
16a	20×10	5	0.1	2266	2310.44	1627.86	2282	2312.02	1655.97	-	2256	2258.0	2271	2303.97	1657.91
17a	20×10	5	0.3	2141	2152.84	716.80	2128	2157.02	819.73	-0.19%	2140	2142.0	2130	2156.86	752.47
18a	20×10	5	0.5	2128	2144.29	489.71	2088	2101.59	714.03	1.99%	2127	2130.7	2091	2103.04	691.46

Table 4.4: Performance comparison for newly generated problems

Problem	Parameters		HGA			PRC			R_{eff}	PRC-TW		
	$ A $	δp_i	Best	Ave.	Time	Best	Ave.	Time		Best	Ave.	Time
p01	1-2	0	864	874.85	1162.06	860	873.34	1128.29	0.17%	864	875.08	1170.25
p02	1-2	10	910	924.33	1313.87	911	923.22	1273.33	-	908	922.21	1271.89
p03	1-2	20	925	937.74	1310.18	925	938.14	1257.96	-	926	937.86	1299.81
p04	1-2	30	850	858.76	1350.35	849	859.67	1270.60	-	849	858.77	1333.19
p05	1-2	40	887	897.49	1283.66	887	894.86	1274.10	0.29%	887	893.90	1304.37
p06	1-2	50	886	899.65	1200.65	882	898.69	1167.38	-	885	898.41	1191.78
p07	1-2	60	912	924.58	1233.89	912	922.79	1200.98	0.19%	909	922.24	1227.82
p08	1-2	70	813	825.10	1341.86	812	825.87	1271.06	-	811	825.27	1339.87
p09	1-2	80	789	808.28	1449.59	793	807.18	1389.83	-	789	807.46	1444.98
p10	1-2	90	698	713.82	1668.37	700	712.68	1578.36	-	701	713.46	1640.28
p11	1-6	0	838	840.82	804.67	838	841.29	788.67	-0.06%	837	841.01	802.21
p12	1-6	10	854	862.95	814.44	847	855.31	803.23	0.89%	846	861.06	821.15
p13	1-6	20	800	811.20	798.42	795	804.82	807.18	0.79%	794	810.51	801.03
p14	1-6	30	769	783.49	929.01	762	772.38	947.42	1.42%	764	776.99	942.10
p15	1-6	40	727	741.69	866.75	725	733.97	851.66	1.04%	720	737.28	870.85
p16	1-6	50	729	742.18	805.30	732	743.68	777.78	-0.20%	734	745.44	783.86
p17	1-6	60	718	729.46	806.20	715	726.65	788.35	0.39%	719	732.52	808.01
p18	1-6	70	586	596.77	1100.69	587	596.25	1062.18	-	591	598.73	1103.18
p19	1-6	80	556	565.60	1145.47	557	565.36	1138.94	-	557	565.87	1126.01
p20	1-6	90	560	568.13	826.32	558	566.50	787.46	0.29%	559	568.09	785.25
p21	1-10	0	883	885.43	562.64	883	885.26	644.42	-	883	885.18	640.94
p22	1-10	10	865	873.51	569.30	854	861.25	633.07	1.40%	854	864.10	628.25
p23	1-10	20	716	727.56	862.22	708	717.06	912.10	1.44%	708	718.69	899.96
p24	1-10	30	747	758.08	765.49	737	747.21	789.26	1.43%	741	751.44	783.19
p25	1-10	40	619	633.04	762.61	614	624.66	770.21	1.32%	616	628.89	773.14
p26	1-10	50	632	645.90	774.54	630	640.16	790.06	0.89%	630	643.10	781.75
p27	1-10	60	555	566.01	879.34	553	563.71	869.82	0.41%	556	566.17	875.64
p28	1-10	70	476	482.66	1051.82	473	480.62	1020.25	0.42%	474	482.23	1038.28
p29	1-10	80	491	501.62	815.09	491	498.96	796.93	0.53%	491	501.45	821.29
p30	1-10	90	518	527.80	764.67	518	530.81	758.06	-0.57%	522	529.77	749.99

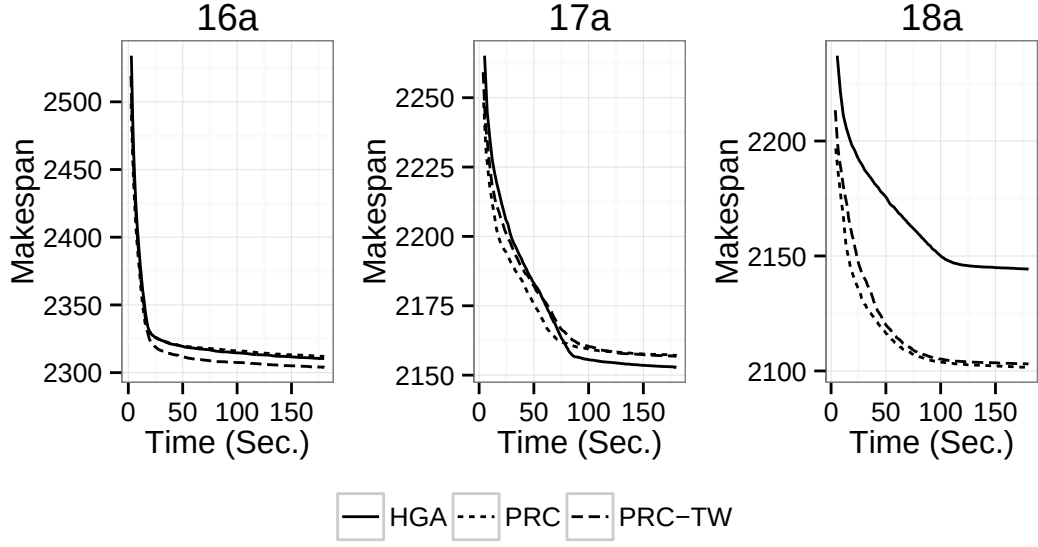


Figure 4.3: Comparison of decreasing speed of makespan for DPdata in long-term

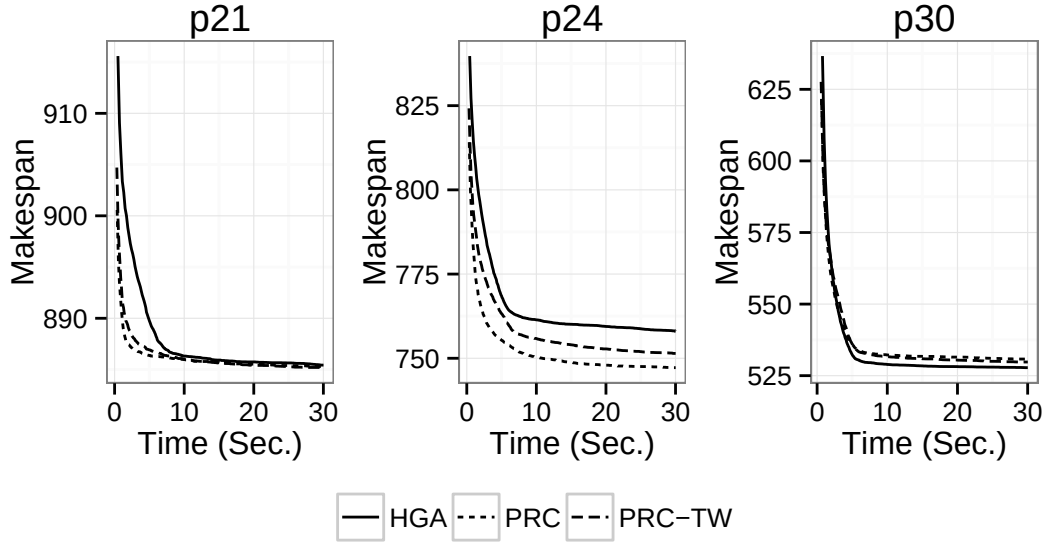


Figure 4.4: Comparison of decreasing speed of makespan for newly generated problems

timal solutions. Problem with large numbers of assignable machines for each operation may contain semi optimal solutions with different machine assignments. When the deviation in processing time on different machines is large, however, it is often preferable to process an operation on the most efficient machine, which is why machine assignments on semi optimal solutions tend to be fixed. Various machine assignments for operations should therefore be examined to obtain a semi optimal schedule for those problems where the

proposed method works well. Although the proposed method uses priority rules to determine both machine assignment and processing sequence on the same machine, these results imply that simultaneous machine reassignments of multiple operations for the proposed method is crucial to effectively solving the problems with the conditions above. In problems with zero deviation in processing time, however, the proposed method was not able to find a suitable machine assignments as the priority rules in the proposed method relies on the difference of processing time of the same operations among different machines.

4.4 Conclusions

In this section, PRC procedure for the FJSP is proposed. This procedure differs from existing algorithms in that priority rules are represented by linearly weighted values and their weights are optimized together with solutions. This procedure is implemented with a well-known algorithm for the FJSP of HGA.

The performance of the proposed method is evaluated by applying it to benchmark problems. It is confirmed that the introduction of a priority rule-based reconstruction improves the quality of solutions under conditions that were investigated with newly generated problems and are described as follows:

- The deviation in processing time on different machines is about 30% of the overall deviation of processing time for all operations.
- The average number of assignable machines is large.

It is also confirmed that the proposed method outperforms a state-of-the-art algorithm on a few problems which meet the above two conditions.

The results showed that types of searches different from the current approach are needed to effectively solve particular problem and implied that switching a strategy based on the property of the problem is crucial for obtaining highly efficient algorithms. Specifically, reassigning machine assignments for multiple operations is effective with problems with the conditions shown

above. It is also revealed that the proposed strategy by the priority rules is better suited for diversification rather than intensification on the FJSP. Future research directions should include the development of an adaptive heuristic approach that switches optimizing strategy based on the type of problem.

Bibliography

- [Bra93] Paolo Brandimarte. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41(3):157–183, 1993.
- [DPP97] Stphane Dauzre-Prs and Jan Paulli. An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research*, 70(0):281–306, 1997.
- [GSG08] Jie Gao, Linyan Sun, and Mitsuo Gen. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers & Operations Research*, 35(9):2892 – 2907, 2008. Part Special Issue: Bio-inspired Methods in Combinatorial Optimization.
- [HJT94] Johann Hurink, Bernd Jurisch, and Monika Thole. Tabu search for the job-shop scheduling problem with multi-purpose machines. *Operations-Research-Spektrum*, 15(4):205–215, 1994.
- [HTL07] Nhu Binh Ho, Joc Cing Tay, and Edmund M.-K. Lai. An effective architecture for learning and evolving flexible job-shop schedules. *European Journal of Operational Research*, 179(2):316 – 333, 2007.
- [MG00] M. Mastrolilli and L. M Gambardella. Effective neighbourhood functions for the flexible job shop problem. *Journal of Scheduling*, 3(1):3–20, 2000.

- [PMC08] F. Pezzella, G. Morganti, and G. Ciaschetti. A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research*, 35(10):3202 – 3212, 2008. Part Special Issue: Search-based Software Engineering.
- [TH08] Joc Cing Tay and Nhu Binh Ho. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering*, 54(3):453 – 473, 2008.
- [TYSF13] Yasumasa Tamura, Masahito Yamamoto, Ikuo Suzuki, and Masashi Furukawa. Acquisition of dispatching rules for job-shop scheduling problem by artificial neural networks using pso. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 17(5):731 – 738, 2013.
- [YX13] Yuan Yuan and Hua Xu. Flexible job shop scheduling using hybrid differential evolution algorithms. *Computers & Industrial Engineering*, 65(2):246 – 260, 2013.
- [ZGS11] Guohui Zhang, Liang Gao, and Yang Shi. An effective genetic algorithm for the flexible job-shop scheduling problem. *Expert Systems with Applications*, 38(4):3563 – 3573, 2011.

Chapter 5

Conclusion

In chapter 1, background and objective of this thesis is introduced. The studies on the classic JSP and extended JSPs are described and relationships with the objectives is explained.

In chapter 2, formulations of two extended JSPs that focused on this thesis, namely the TJSP and the FJSP, is given. The difficulties of the extended JSPs compared to the classic JSP is explained as well as current status of the studies of approximation algorithms for them. Finally differences between the TJSP and FJSP is discussed and it is implied that different approaches of priority rules are needed for them.

In chapter 3, a priority rule-based heuristic is integrated to an existing approximation algorithm for the TJSP. Local search procedure is augmented by a priority rule-based heuristic because additional indexes for priority rules are available for the TJSP and thus they are relatively more efficient for them. The proposed reconstruction procedure receives a solution which is possibly local optima, time window, and composition parameters for each index. Then, the operation sequences of the solution inside the time window is reconfigured by the composed priority rule to construct another solution. To obtain better solution with this procedure, time window and composition parameters are optimized by genetic algorithm. This reconstruction procedure is performed on a solution acquired by the existing local search algorithm to help escap-

ing from local optima. To evaluate the efficiency of the priority rule-based heuristics, the proposed method is performed on benchmark problems and the results are compared to that of existing algorithms. From the result, it is confirmed that priority rule-based heuristics can improve the existing algorithms. It is also revealed that they are particularly efficient on the problem where the number of jobs are larger than that of machines.

In chapter 4, a priority rule-based heuristics is applied to the FJSP. Priority rule-based reconstruction procedure is utilized to limit extended search space introduced by the FJSP. The proposed reconstruction procedure is similar to that of the TJSP except for that time window is not employed in order to find a promising region from the extended search space. The procedure is then integrated to the existing HGA which perform local search procedures on each chromosome each generation. The proposed method is performed on benchmark problems and the results are compared to that of existing algorithms to evaluate its efficiency. As combined results which shows that the proposed problems performs better on some problems and performs worse on the other problems contained in the same benchmark set, they are performed on randomly generated problems to further investigate the performance of the proposed algorithm. As a result, it is shown that the problems whose operations can be processed on large number of machines tends to be solved effectively by the proposed method. This implies that the intention of restricting search space is valid as such problems should have larger search space.

Finally, we summarize this thesis with some remarks and gives some directions toward the future work. In this thesis, priority rule-based method is applied to the two extended JSPs and its efficiency is confirmed. This result mean priority rule-based method can be effective on the extended JSPs and thus it is promising on other extended JSPs that is not investigated in this thesis. Other than that, there are studies on a general priority rule which is efficient on every problem. Integrating meta-heuristic approaches for the JSP with such rule is advantageous in that they can be pre-obtained and computation efforts needed to acquire optimized priority rules for each problem is

not required. Therefore, utilizing general rules in meta-heuristic approaches for the JSP might be interesting.

Acknowledgment

The author express deep gratitude to Professor Masahito Yamamoto of Hokkaido University for constant polite instruction and guidance from the inception to the completion of the present research.

Deep thanks are also due to Professor Masahito Kurihara, Professor Tetsuo Ono, Professor Keiji Suzuki, Associate Professor Hiroyuki Iizuka of Hokkaido University.

He would like to thank Professor Masashi Furukawa of Hokkaido Information University and Associate Professor Ikuo Suzuki of Kitami Institute of Technology for profitable discussion as well as intruction and guidance in Hokkaido University.

Many pieces of affectionate advice and discussion given by them were very useful to improve this research.

He would like to thank all students of the office, Autonomous Systems Engineering Laboratory, Synergetic Information Engineering. The conversations between them always encouraged him. Particularly Yasumasa Tamura, Jun Ogawa, Masaya Honjo, and Yukihiro Tagawa were always of great help to him.

Finally, the author would like to thank to his family; his father Soichiro Yokoyama, his mother Kyoko Yokoayama, his sister Yuko Yokoyama, his brother Keijiro Yokoyama, and his brother Junzaburo Yokoyama. Without his family, he would not have continued his research.

This work was supported by JSPS KAKENHI Grant Number 15J01601.

Bibliography

- [Bie95] Christian Bierwirth. A generalized permutation approach to job shop scheduling with genetic algorithms. *Operations-Research-Spektrum*, 17(2-3):87–92, 1995.
- [Bon05] K.M.J. Bontridder. Minimizing total weighted tardiness in a generalized job shop. *Journal of Scheduling*, 8(6):479–496, 2005.
- [Bra93] Paolo Brandimarte. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41(3):157–183, 1993.
- [BV98] Egon Balas and Alkis Vazacopoulos. Guided local search with shifting bottleneck for job shop scheduling. *Manage. Sci.*, 44(2):262–275, February 1998.
- [DP95] Ulrich Dorndorf and Erwin Pesch. Evolution based learning in a job shop scheduling environment. *Computers & Operations Research*, 22(1):25 – 40, 1995. Genetic Algorithms.
- [DPP97] Stphane Dauzre-Prs and Jan Paulli. An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research*, 70(0):281–306, 1997.
- [EBS06] Ahmed El-Bouri and Pramit Shah. A neural network for dispatching rule selection in a job shop. *The International Journal of Advanced Manufacturing Technology*, 31(3-4):342–349, 2006.

- [EMDP08] Imen Essafi, Yazid Mati, and Stéphane Dauzère-Pérès. A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. *Comput. Oper. Res.*, 35(8):2599–2616, August 2008.
- [FB91] E. Falkenauer and S. Bouffouix. A genetic algorithm for job shop. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 824–829 vol.1, Apr 1991.
- [GSG08] Jie Gao, Linyan Sun, and Mitsuo Gen. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers & Operations Research*, 35(9):2892 – 2907, 2008. Part Special Issue: Bio-inspired Methods in Combinatorial Optimization.
- [GT60] B. Giffler and G. L. Thompson. Algorithms for solving production-scheduling problems. *Operations Research*, 8(4):pp. 487–503, 1960.
- [GZZL11] Liang Gao, Guohui Zhang, Liping Zhang, and Xinyu Li. An efficient memetic algorithm for solving the job shop scheduling problem. *Computers & Industrial Engineering*, 60(4):699 – 705, 2011.
- [HJT94] Johann Hurink, Bernd Jurisch, and Monika Thole. Tabu search for the job-shop scheduling problem with multi-purpose machines. *Operations-Research-Spektrum*, 15(4):205–215, 1994.
- [HSEC09] S.M.Kamrul Hasan, Ruhul Sarker, Daryl Essam, and David Cornforth. Memetic algorithms for solving job-shop scheduling problems. *Memetic Computing*, 1(1):69–83, 2009.
- [HTL07] Nhu Binh Ho, Joc Cing Tay, and Edmund M.-K. Lai. An effective architecture for learning and evolving flexible job-shop schedules. *European Journal of Operational Research*, 179(2):316 – 333, 2007.

- [Kre00] Stephan Kreipl. A large step random walk for minimizing total weighted tardiness in a job shop. *Journal of Scheduling*, 3(3):125–138, 2000.
- [LYL98] Kyung-Mi Lee, T. Yamakawa, and Keon-Myung Lee. A genetic algorithm for general machine scheduling problems. In *Knowledge-Based Intelligent Electronic Systems, 1998. Proceedings KES '98. 1998 Second International Conference on*, volume 2, pages 60–66 vol.2, Apr 1998.
- [MFDP⁺11] Lars Mönnch, John W. Fowler, Stéphane Dauzère-Pérès, Scott J. Mason, and Oliver Rose. A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations. *Journal of Scheduling*, 14(6):583–599, 2011.
- [MG00] M. Mastrolilli and L. M Gambardella. Effective neighbourhood functions for the flexible job shop problem. *Journal of Scheduling*, 3(1):3–20, 2000.
- [NS96] Eugeniusz Nowicki and Czeslaw Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42(6):797–813, 1996.
- [NY91] Ryohei Nakano and Takeshi Yamada. A genetic algorithm for job shop. In *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 474–479, Jan 1991.
- [NZJT13] Su Nguyen, Mengjie Zhang, M. Johnston, and Kay Chen Tan. A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem. *Evolutionary Computation, IEEE Transactions on*, 17(5):621–639, Oct 2013.
- [PI77] S. S. Panwalkar and Wafik Iskander. A survey of scheduling rules. *Operations Research*, 25(1):45–61, 1977.

- [PMC08] F. Pezzella, G. Morganti, and G. Ciaschetti. A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research*, 35(10):3202 – 3212, 2008. Part Special Issue: Search-based Software Engineering.
- [PS99] Michael Pinedo and Marcos Singer. A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop. *Naval Research Logistics (NRL)*, 46(1):1–17, 1999.
- [pWHW05] Jean paul Watson, Adele E. Howe, and L. Darrell Whitley. Deconstructing nowicki and smutnicki’s i-tsab tabu search algorithm for the job-shop scheduling problem. *Computers & Operations Research*, 33:2623–2644, 2005.
- [QW12] Ren Qing-dao-er-ji and Yuping Wang. A new hybrid genetic algorithm for job shop scheduling problem. *Computers & Operations Research*, 39(10):2291 – 2299, 2012.
- [SP98] Marcos Singer and Michael Pinedo. A computational study of branch and bound techniques for minimizing the total weighted tardiness in job shops. *IIE Transactions*, 30(2):109–118, 1998.
- [Tai94] Eric D. Taillard. Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing*, 6(2):108–117, 1994.
- [TH08] Joc Cing Tay and Nhu Binh Ho. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering*, 54(3):453 – 473, 2008.
- [TYSF13] Yasumasa Tamura, Masahito Yamamoto, Ikuo Suzuki, and Masashi Furukawa. Acquisition of dispatching rules for job-shop scheduling problem by artificial neural networks using pso. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 17(5):731 – 738, 2013.

- [vLAL92] Peter J. M. van Laarhoven, Emile H. L. Aarts, and Jan Karel Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40(1):113–125, 1992.
- [VM87] Ari P. J. Vepsalainen and Thomas E. Morton. Priority rules for job shops with weighted tardiness costs. *Management Science*, 33(8):pp. 1035–1047, 1987.
- [YX13] Yuan Yuan and Hua Xu. Flexible job shop scheduling using hybrid differential evolution algorithms. *Computers & Industrial Engineering*, 65(2):246 – 260, 2013.
- [ZBH05] Jeriad Zoghby, J. Wesley Barnes, and John J. Hasenbein. Modeling the reentrant job shop scheduling problem with setups for metaheuristic searches. *European Journal of Operational Research*, 167(2):336 – 348, 2005.
- [ZCL09] Hong Zhou, Waiman Cheung, and Lawrence C. Leung. Minimizing weighted tardiness of job-shop scheduling using a hybrid genetic algorithm. *European Journal of Operational Research*, 194(3):637 – 649, 2009.
- [ZGS11] Guohui Zhang, Liang Gao, and Yang Shi. An effective genetic algorithm for the flexible job-shop scheduling problem. *Expert Systems with Applications*, 38(4):3563 – 3573, 2011.
- [ZLGR07] ChaoYong Zhang, PeiGen Li, ZaiLin Guan, and YunQing Rao. A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. *Computers & Operations Research*, 34(11):3229 – 3242, 2007.
- [ZSW13] Rui Zhang, Shiji Song, and Cheng Wu. A dispatching rule-based hybrid genetic algorithm focusing on non-delay schedules for the job shop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, 67(1-4):5–17, 2013.