Journal of Computer Engineering
ISSN 10003428

Original Research Article
Vol 9, Issue 4, 2020 | ISSN: 1000-3428

Optimization Strategies in Job Shop Scheduling: A Comprehensive Analysis

Dr. Leila Jafari, Dr. Muhammad Khalid

Department of Industrial Engineering, University of Tehran, Tehran, Iran; School of Engineering and Information Technology, Murdoch University, Perth, Australia

***Abstract*—**The job-shop scheduling problem (JSSP) is an important decision facing those involved in the fields of industry, economics and management. This problem is a class of combinational optimization problem known as the NP-hard problem. JSSPs deal with a set of machines and a set of jobs with various predetermined routes through the machines, where the objective is to assemble a schedule of jobs that minimizes certain criteria such as makespan, maximum lateness, and total weighted tardiness. Over the past several decades, interest in meta-heuristic approaches to address JSSPs has increased due to the ability of these approaches to generate solutions which are better than those generated from heuristics alone. This article provides the classification, constraints and objective functions imposed on JSSPs that are available in the literature.

I. Introduction

THE ability to make timely, effective decisions is one of the most important issues faced in manufacturing; slow or poor decisions increase production costs. Thus, good and timely decisions based on optimal scheduling of the production process with regards to limited available resources constitute a key factor in the efficient control of production. The JSSP is a NP-hard problem [1]. From the mid-50s onwards, many researchers have been interested in expanding the theoretical models of the JSSP and have introduced algorithms to solve them. Among these, some have tried to review, categorize and analyse the various methodologies applied to the JSSP. The fundamental work that first reviewed the proposed methodologies for the JSSP was by [2]. Later, [3] revisited the JSSP as one of the production scheduling problems. Nearly two decades later, [4] reviewed exact methods and hybrid techniques as iterated local search algorithms. They developed guidelines on the features that should be included to make a suitable job-shop scheduling system. Thereafter, several other researchers reviewed and analysed available methodologies for the JSSP. This paper aims to pay particular attention on a review on the classification, constraints and objective functions of the JSSP that are available in the literature.

The remainder of the paper is organized as follows: Section II presents the classification of JSSPs. Section III describes the
various constraints and objective functions imposed on JSSPs followed by a brief summary in section IV.

II. Classification of JSSP

Although JSSPs have overlapping characteristics, they can be classified with respect to several facets. The job arrival process, the inventory policy, duration time processing and job attributes are the most widely used among those facets. Based on the literature, there are 14 classes of JSSP (Fig. 1): Deterministic JSSP, flexible JSSP, static JSSP, dynamic JSSP, periodic JSSP, cyclic JSSP, pre-emptive JSSP, no-wait JSSP, just-in-time JSSP, large-scale JSSP, re-entrant JSSP, assembly JSSP, stochastic JSSP, and fuzzy JSSP.

The **deterministic/crisp JSSP** [5] is a popular type of JSSP. The deterministic JSSP consists of n jobs to be processed on m machines. Each job should be processed on all machines and consists of a chain or complex of operations, which have to be scheduled in a predetermined given order. Each operation has to be processed on a given machine for an uninterrupted processing time period and no operation may be pre-empted. The deterministic JSSP occurs when the operations of jobs, processing times of operations and availability of machines are given by a crisp value. This JSSP is different from scheduling problems under uncertainty such as the stochastic JSSP and fuzzy JSSP that have imprecise parameters.

The **flexible JSSP** is an extension of the crisp JSSP and it allows an operation to be processed by one machine out of a set of candidate machines. The problem consists of two subproblems: assigning each operation to a machine (routing problem) and sequencing the assigned operations on the machines (sequencing problem), such that a feasible schedule optimization is achieved for given objectives. So, the flexible JSSP includes an extra problem; the assignment of operations to machines. The flexible JSSP has recently gained the attention of many researchers [6]-[8].

JSSPs based on the job arrival process are classified into two categories: the **static JSSP** and the **dynamic JSSP**. In the **static JSSP** [9], a finite set of jobs, including a predefined order of precedence of operations for each job, needs to be processed on a finite set of machines. At the beginning of the static JSSP, all jobs are released and all machines are available. Each machine can process only one operation at a time, and each job cannot be operated simultaneously by more than one machine. The problem does not have any unexpected events or machine breakdown during the scheduling process. It should be

How to cite: Dr. Leila Jafari, Dr. Muhammad Khalid (2020). Optimization Strategies in Job Shop Scheduling: A Comprehensive Analysis. Journal of Computer Engineering, 9(4), 13–22. DOI: https://doi.org/10.5281/zenodo.19341543

noted that deterministic JSSP and static JSSP differ from each other; if the JSSP classification is based on time parameters, then deterministic, fuzzy and stochastic JSSPs can be created, whereas if the JSSP categories based on the job arrival process, then static and dynamic JSSPs can be obtained.

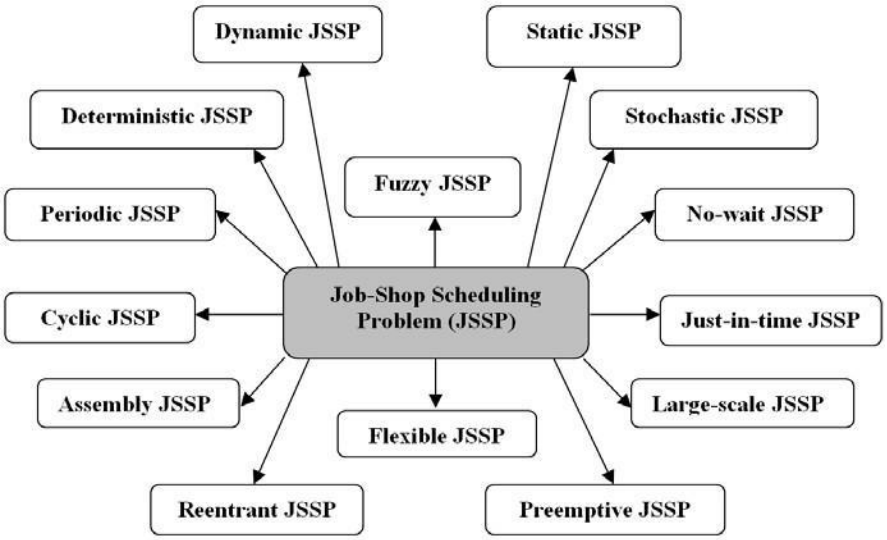

Fig. 1 Fourteen classes of JSSP

The **dynamic JSSP** contains a set of machines and jobs of various types that arrive continuously over time in a random manner. Each job consists of a specific set of operations that should be performed in a specified order (routing) on the machines and involves a certain amount of processing time. In the deterministic JSSP, a job leaves one machine and proceeds on its route to another machine for the next operation only to find other jobs already waiting for the machine to complete its current task [10]; thus, the deterministic JSSP becomes a queuing system. The dynamic JSSP essentially involves deciding the order or priority of the jobs waiting to be processed by each machine to achieve the desired objectives. Scheduling rules or dispatching rules are applied for this purpose. These rules are accepted in industry because of their ease of implementation, satisfactory performance, low computation requirement, and flexibility to incorporate domain knowledge and expertise [10]. The dynamic JSSP has also recently has attracted the interest of many researchers [11]-[13].

The **periodic JSSP** is employed to schedule a finite set of jobs in each single working shift, where all shifts have similar operations, as well as the same scheduling. The periodic JSSP assigns operations to time slots with the maximum length of *T*, on corresponding machines. In this problem, some periodic events are repeated subject to certain constraints. Also, every constraint in a given set of periodic interval constraints is associated with a pair of events confined by a lower bound and an upper bound [14], [15].

The **cyclic JSSP** [16], [17] consists of a set of operations that are to be repeated in a process an indefinite number of times. In this problem, the primary objective function is to minimize the period length. The cyclic JSSP occurs in domains in which the sequences of jobs are repeated. The problem is generally studied in two different ways: with and without resource constraints [18].

The **pre-emptive JSSP** [19] includes pre-emption constraints. These constraints imply that:

> *"...it is not necessary to keep a job on a machine, once started, until its completion. The scheduler is allowed to interrupt the processing of [a] job (pre-empt) at any point in time and put a different job on the machine instead. The amount of processing a pre-empted job already has received is not lost. When a pre-empted job is afterwards put back on the machine (or on another machine in the case of parallel machines), it only needs the machine for its remaining processing time"* [5: p. 16].

The **no-wait JSSP** [20] applies to the following situation:

> *"Jobs are not allowed to wait between two successive machines. This implies that the starting time of a job at the first machine has to be delayed to ensure that the job can go through the flow shop without having to wait for any machine*" [5: p. 17].

The **just-in-time JSSP** [21] is also devoted to the solving the earliness-tardiness type of JSSP because both the early completion time of jobs (which results in the requirement for storage) and the tardy completion time of jobs are penalized. In this problem, a due date for each operation and two penalty coefficients to penalize the early or tardy completion are considered.

The **large-scale JSSP** [22] implies the existence of a large number of machines and number of jobs. Due to this problem having a too huge solution space and the need to design special algorithms to solve it, this problem is considered to be an independent category of JSSP.

The **re-entrant JSSP** [23] is an extension of the classic crisp JSSP, where jobs may visit machines more than once, i.e., a job may have two or more operations that require the same machine. The re-entrant JSSP can be summarized as follows: "*Each job may process with certain machines more than once. Any two consecutive operations of each job cannot be processed on the same machine. No two machines are allowed to perform the same operation. The processing times are independent of the sequence. There is no randomness; all the data are known and fixed. All jobs are ready for processing at time zero at which the machines are idle and immediately available for work. No pre-emption is allowed, i.e., once an operation is started, it must be completed before another operation can be started on that machine. Machines never breakdown and are available throughout the scheduling period. The technological constraints are known in advance and immutable. There is only one of each type of machine. Inprocess inventory is allowed*" [24: p. 1198].

The **assembly JSSP** [25] appends an assembly relationship and lot splitting for the final production. The problem can be explained as follows: "*A job is regarded as 'completed' if all of its operations are finished and it implies that each job is independent. Usually, a job is a batch of identical items in which the whole batch cannot be split. This means that the batch must be wholly transferred from machine to machine even some items have already been processed. The first restriction assumes that each job is independent and the second assumes that each job cannot be split. In reality, these two restrictions are not always valid. To relax the first restriction, an assembly stage is attached such that JSSP becomes an Assembly JSSP. In this connection, each job is an entity of the Bill-Of-Material (BOM) of all products. The assembly relationship between different jobs is defined by the BOMs. If there is no common part, only jobs from the same BOM can be assembled. In contrast, only common parts from distinct BOMs may be assembled*" [26: p. 983].

The **stochastic JSSP** [27] consists of parameters that are initially described in terms of probability distributions. This problem is an important aspect to address in respect of manufacturing systems and the extended version of JSSP by introducing some stochastic processing conditions such as the probability of machine breakdown, as well as stochastic processing time in normal exponential and uniform distributions [28].

The assumption that the duration times of JSSPs in the realworld have a crisp value is often violated in practice. Since human-centred factors are incorporated into the JSSP, it may be more appropriate to consider a fuzzy processing time due to human-made factors and a fuzzy due date which tolerates a certain amount of delay in the due date. Therefore, another newer type of JSSP, the **fuzzy JSSP** [29] has come to prominence. The basic characteristics of a fuzzy JSSP are fuzzy processing time, fuzzy due date, and fuzzy ranking (fuzzy max). Also, the fuzzy JSSP has three subclasses: fuzzy JSSP with fuzzy due date, fuzzy JSSP with fuzzy processing time, and fuzzy JSSP with both fuzzy processing time and fuzzy due date.

III. Constraints and Objective Functions

In the classical JSSP [30], there are set of jobs on a set of machines. The operations of the jobs have to be processed on the machines a set of constraints. Fig. 2 presents the classic constraints, extra constraints and the objective functions of the JSSPs that have been studied in the literature.

A. Classic Constraints of JSSPs

There are three types of constraints: **precedence**, **capacity**, and **release and due date**. Precedence constraints include three limitations; each job should be processed through the sequence of machines in a predetermined order, the machine orders among different jobs are unconfined and there are no precedence constraints among the operations of different jobs. Capacity constraints comprise five restraints; machines are independent of one another, machines cannot remain idle while an operation is waiting for processing, each machine can only mostly handle one operation at a time, each job can be processed only once on a given machine, and jobs are independent of each other. Finally, release and due date constraints contain three restrictions; there is no negative starting time, the processing time of operations is given a length, and the processing of each operation must not be interrupted. Therefore, to satisfy these constraints and to achieve the objective of a JSSP, the starting time of the processing operation is considered to be the decision variable of the JSSP. A schedule is assigned time slots on the machines for operations by satisfying the problem's constraints and finding a sequence of jobs on the machines; the corresponding schedule should optimize the objective function of the JSSP. The above classic constraints are the formal constraints that are considered for the 14 classes of JSSPs discussed in the previous section; however, there are some extra constraints that are considered for the deterministic JSSP (crisp JSSP) and in other types of JSSP.

B. Extra Constraints of JSSPs

The **JSSP with multipurpose machines** [31] or the **JSSP with unrelated parallel constraints** [32] associates each operation with a subset of machines, from which exactly one must be chosen to process the operation.

Recall that in the flexible JSSP, the operations may have different processing times on different machines.

The **JSSP with processing alternative** [33] is an extended JSSP that considers machine alternatives for individual operations and allows jobs to have a partial order of operations. The extension evolved from JSSP with multipurpose machines or flexible JSSP into a multimode JSSP [31] for multi-resource shop scheduling with resource flexibility [34].

The **JSSP with deteriorating jobs** [35] means the processing times of jobs are an increasing function of their starting time. In most of the research related to scheduling deteriorating jobs, a simple linear deterioration function is assumed.

The **JSSP with maintenance activity** [36] implies that each machine is subject to preventive maintenance during the planning period. The starting times of maintenance activities are either flexible in a time window or fixed beforehand. Moreover, two cases of maintenance resource constraint are considered: sufficient maintenance resource being available or only one maintenance resource being available.

The **JSSP with sequence-dependent setup times** [37] or **JSSP with setup times** [38] is a JSSP where the magnitude of the setup strongly depends on both the current and the immediately processed jobs on a given machine. For example, this may occur in a painting operation, where different initial paint colours require different levels of preparation before being painted over with other paint colours.

The **JSSP with availability constraints** [39] implies that one or several machines might cease to be available for processing jobs after a breakdown or when preventive maintenance such as washing or control operations is scheduled.

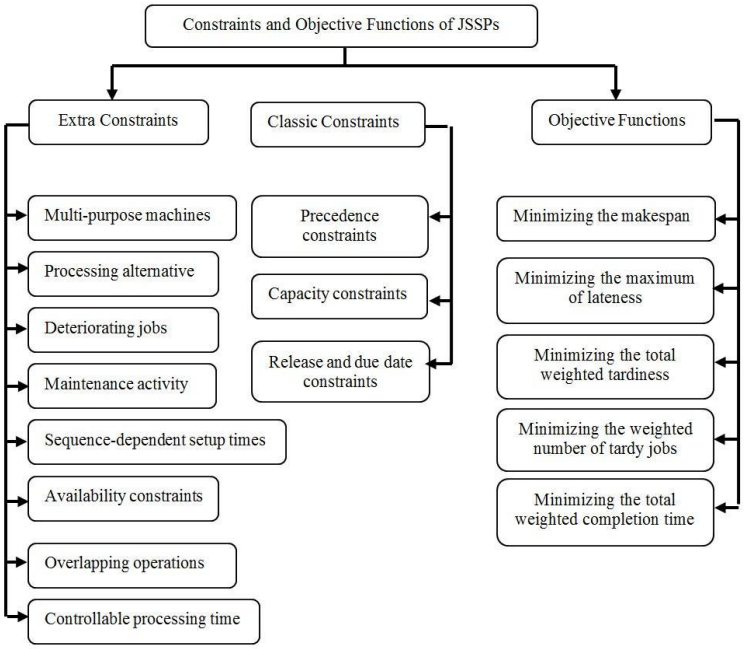

Fig. 2 Constraints and objective functions of JSSPs

The **JSSP with overlapping operations** [40] includes a demand with a value of more than 1 for each job. The demand determines the quantity of each completed job ordered by a customer. Based on this assumption, the operations of each job can be performed based on overlap considerations, where an operation may be overlapped with others because of its nature. The overlapping is limited by structural constraints, such as the dimensions of the box to be packed or the capacity of the container used to move the pieces from one machine to the next.

The **JSSP with controllable processing time** [41] implies that the processing time of a job can be reduced because additional resources are assigned to the job. There are situations where it is possible to compress a job, but this entails an extra cost. These situations are acceptable only if the additional cost is compensated by the gains from job completion at an earlier time.

C. Objective Functions of JSSPs

Minimizing the makespan [42]. In this case, "the makespan is equivalent to the completion time of the last job to leave the system. A minimum makespan usually implies a good utilization of the machine(s)" [5: p. 18].

Minimizing the total weighted completion time [43]. Here "the sum of the weighted completion times of the n jobs gives an indication of the total holding or inventory costs incurred by the schedule. The sum of the completion times is in the literature often referred to as the flow time. The total weighted completion time is then referred to as the weighted flow time" [5: p. 19].

Minimizing the maximum of lateness [44]. In this case, because the objective is a function of the due date, the lateness of job i is defined as $L_i=C_i-d_i$, where C_i and d_i are the completion time and due date of job i, respectively. The lateness of job i is positive when job i is completed late and negative when it is completed early. The maximum of lateness calculates the worst violation of the due dates.

Minimizing the total weighted tardiness [45]. The second objective related to due date is tardiness. The tardiness of job i is defined as $T_i=\max(C_i-d_i,0)=\max(L_i,0)$. "The difference between the tardiness and the lateness lies in the fact that the tardiness never is negative" [5: p. 18].

This objective "is also a more general cost function than the total weighted completion time" [5: p. 19].

Minimizing the weighted number of tardy jobs [46] or **minimizing the weighted number of late jobs** [47] is the third objective related to due date. The unit penalty of job i or the tardy job i is defined as $U_i=1$ if $C_i>d_i$, else $U_i=0$. So the weighted number of tardy jobs is calculated by $\sum w_i U_i$. This objective "is not only a measure of academic interest, it is often an objective in practice as it is a measure that can be recorded very easily" [5: p. 19].

IV. Conclusions

This paper reviewed a number of works in the domain of JSSPs

References

[1] Lageweg, B., Lenstra, J., & Kan, A. H. G. R. (1977). Job-shop scheduling by implicit enumeration. Management Science, 441-450.

[2] Mellor, P. (1966). A review of job shop scheduling. Operations Research, 161-171.

[3] Graves, S. C. (1981). A review of production scheduling. Operations Research, 29(4), 646-675.

[4] Jain, A. S., & Meeran, S. (1999). Deterministic job-shop scheduling: Past, present and future. European Journal of Operational Research, 113(2), 390-434.

[5] Pinedo, M. L. (Ed.). (2012). Scheduling, Theory, Algorithm and Systems. New York: Springer.

[6] Wang, L., Zhou, G., Xu, Y., Wang, S., & Liu, M. (2012). An effective artificial bee colony algorithm for the flexible job-shop scheduling problem. The International Journal of Advanced Manufacturing Technology, 60(1), 303-315.

[7] Moslehi, G., & Mahnam, M. (2011). A Pareto approach to multiobjective flexible job-shop scheduling problem using particle swarm optimization and local search. International Journal of Production Economics, 129(1), 14-22.

[8] Zhang, G., Gao, L., & Shi, Y. (2011). An effective genetic algorithm for the flexible job-shop scheduling problem. Expert Systems with Applications, 38(4), 3563-3573.

[9] Qiu, X., & Lau, H. Y. (2014). An AIS-based hybrid algorithm for static job shop scheduling problem. Journal of Intelligent Manufacturing, 25(3), 489-503.

[10] Vinod, V., & Sridharan, R. (2011). Simulation modeling and analysis of due-date assignment methods and scheduling decision rules in a dynamic job shop production system. International Journal of Production Economics, 129(1), 127-146.

[11] Adibi, M. A., & Shahrabi, J. (2014). A clustering-based modified variable neighborhood search algorithm for a dynamic job shop scheduling problem. The International Journal of Advanced Manufacturing Technology, 70(9-12), 1955-1961.

[12] Zhang, L., Gao, L., & Li, X. (2013). A hybrid genetic algorithm and tabu search for a multi-objective dynamic job shop scheduling problem. International Journal of Production Research, 51(12), 3516-3531.

[13] Nie, L., Gao, L., Li, P., & Li, X. (2012). A GEP-based reactive scheduling policies constructing approach for dynamic flexible job shop scheduling problem with job release dates. Journal of intelligent Manufacturing, 1-12.

[14] Ahmad, F., & Khan, S. A. (2012). Module-based architecture for a periodic job-shop scheduling problem. Computers & Mathematics with Applications, 64(1), 1-10.

[15] Jamili, A., Shafia, M., & Tavakkoli-Moghaddam, R. (2011(a)). A hybridization of simulated annealing and electromagnetism-like mechanism for a periodic job shop scheduling problem. Expert Systems with Applications, 38(5), 5895-5901.

[16] Brucker, P., Burke, E. K., & Groenemeyer, S. (2012a). A mixed integer programming model for the cyclic job-shop problem with transportation. Discrete applied mathematics, 160(13-14), 1924-1935.

[17] Brucker, P., Burke, E. K., & Groenemeyer, S. (2012b). A branch and bound algorithm for the cyclic job-shop problem with transportation. Computers & Operations Research, 39(12), 3200-3214.

[18] Brucker, P., & Kampmeyer, T. (2008). A general model for cyclic machine scheduling problems. Discrete applied mathematics, 156(13), 2561-2572.

[19] Ebadi, A., & Moslehi, G. (2012). Mathematical models for preemptive shop scheduling problems. Computers & Operations Research, 39(7), 1605-1614.

[20] Schuster, C. J., & Framinan, J. M. (2003). Approximative procedures for no-wait job shop scheduling. Operations Research Letters, 31(4), 308318.

[21] Baptiste, P., Flamini, M., & Sourd, F. (2008). Lagrangian bounds for just-in-time job-shop scheduling. Computers & Operations Research, 35(3), 906-915.

[22] Zhang, R., & Wu, C. (2010). A hybrid approach to large-scale job shop scheduling. Applied intelligence, 32(1), 47-59.

[23] Topaloglu, S., & Kilincli, G. (2009). A modified shifting bottleneck heuristic for the reentrant job shop scheduling problem with makespan minimization. The International Journal of Advanced Manufacturing Technology, 44(7), 781-794.

[24] Pan, J. C. H., & Chen, J. S. (2005). Mixed binary integer programming formulations for the reentrant job shop scheduling problem. Computers & Operations Research, 32(5), 1197-1212.

[25] Wong, T., & Ngan, S. (2013). A comparison of hybrid genetic algorithm and hybrid particle swarm optimization to minimize makespan for assembly job shop. Applied Soft Computing, 13(3), 1391-1399.

[26] Wong, T., Chan, F. T. S., & Chan, L. (2009). A resource-constrained assembly job shop scheduling problem with Lot Streaming technique.
Computers & industrial engineering, 57(3), 983-995.

[27] Gu, J., Gu, M., Cao, C., & Gu, X. (2010). A novel competitive coevolutionary quantum genetic algorithm for stochastic job shop scheduling problem. Computers & Operations Research, 37(5), 927-937.

[28] Lei, D. (2012). Co-evolutionary genetic algorithm for fuzzy flexible job shop scheduling. Applied Soft Computing, 12(8), 2237-2245.

[29] Kuroda, M., & Wang, Z. (1996). Fuzzy job shop scheduling.
International Journal of Production Economics, 44(1), 45-51.

[30] French, S. (1982). Sequencing and scheduling: an introduction to the mathematics of the job-shop: Ellis Horwood Chichester.

[31] Brucker, P., & Schlie, R. (1990). Job-shop scheduling with multipurpose machines. Computing, 45(4), 369-375.

[32] Tavakkoli-Moghaddam, R., Taheri, F., Bazzazi, M., Izadi, M., & Sassani, F. (2009). Design of a genetic algorithm for bi-objective unrelated parallel machines scheduling with sequence-dependent setup times and precedence constraints. Computers & Operations Research, 36(12), 3224-3230.

[33] Kis, T. (2003). Job-shop scheduling with processing alternatives. European Journal of Operational Research, 151(2), 307-332.
[34] Dauzere-Peres, S., Roux, W., & Lasserre, J. (1998). Multi-resource shop scheduling with resource flexibility. European Journal of Operational Research, 107(2), 289-305.
[35] Wang, J. B., & Xia, Z. Q. (2005). Scheduling jobs under decreasing linear deterioration. Information Processing Letters, 94(2), 63-69.
[36] Li, J. Q., & Pan, Q. (2012). Chemical-reaction optimization for flexible job-shop scheduling problems with maintenance activity. Applied Soft Computing, 12(9), 2896-2912.
[37] Bagheri, A., & Zandieh, M. (2011). Bi-criteria flexible job-shop scheduling with sequence-dependent setup times—Variable neighborhood search approach. Journal of Manufacturing Systems, 30(1), 8-15.
[38] Balas, E., Simonetti, N., & Vazacopoulos, A. (2008). Job shop scheduling with setup times, deadlines and precedence constraints. Journal of Scheduling, 11(4), 253-262.
[39] Zribi, N., El Kamel, A., & Borne, P. (2008). Minimizing the makespan for the MPM job-shop with availability constraints. International Journal of Production Economics, 112(1), 151-160.
[40] Fattahi, P., & Saidi, M. M. (2009(a)). A New Approach in Job Shop Scheduling: Overlapping Operation. Journal of Industrial Engineering.
[41] Jansen, K., Mastrolilli, M., & Solis-Oba, R. (2005). Approximation schemes for job shop scheduling problems with controllable processing times. European Journal of Operational Research, 167(2), 297-319.
[42] Abdolrazzagh-Nezhad, M., & Abdullah, S. (2014). A Robust Intelligent Construction Procedure for Job-Shop Scheduling. Information Technology And Control, 43(3), 217-229.
[43] Schulz, A. (1996). Scheduling to minimize total weighted completion time: Performance guarantees of LP-based heuristics and lower bounds. Integer Programming and Combinatorial Optimization, 301-315.
[44] McMahon, G., & Florian, M. (1975). On scheduling with ready times and due dates to minimize maximum lateness. Operations Research, 23(3), 475-482.
[45] Zhang, R., & Wu, C. (2011). A simulated annealing algorithm based on block properties for the job shop scheduling problem with total weighted tardiness objective. Computers & Operations Research, 38(5), 854-867.
[46] Chiang, T.-C., & Fu, L.-C. (2008). A rule-centric memetic algorithm to minimize the number of tardy jobs in the job shop. International Journal of Production Research, 46(24), 6913-6931.
[47] Sevaux, M., & Dauzere-Peres, S. (2003). Genetic algorithms to minimize the weighted number of late jobs on a single machine. European Journal of Operational Research, 151(2), 296-306.