Job Shop Scheduling: Classification, Constraints and Objective Functions

Majid Abdolrazzaghi-Nezhad, Salwani Abdullah

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.

Keywords—Job-shop scheduling, classification, constraints, objective functions.

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 sub-problems: 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 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.

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 JSP, 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 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 JSP 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 JSP 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 JSP 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 JSP, where jobs may visit machines more than once,

---

**Fig. 1 Fourteen classes of JSP**
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. In-process 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 real-world 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 sets 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 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.

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 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. This review was conducted to bring together for ease of reference information on the classification, constraints and objectives of JSSPs that have been applied so far.

Acknowledgement

This work was supported by the Ministry of Higher Education, Malaysia FRGS/1/2015/ICT02/UKM/01/2) and the Universiti Kebangsaan Malaysia (DIP-2012-15 and GP-K008253).

References