Computers & Operations Research 74 (2016) 31–41

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Parallel-identical-machine job-shop scheduling with different stage-dependent buffering requirements

Shi Qiang Liu, Erhan Kozan *

School of Mathematical Sciences, Queensland University of Technology, Brisbane, QLD 4001, Australia

ARTICLE INFO

Article history:
Received 6 March 2014
Received in revised form
12 November 2015
Accepted 25 April 2016
Available online 26 April 2016

Keywords:
Job shop scheduling
Parallel machine
Blocking
No-wait
Limited-buffer
Mixed integer programming
Heuristics

ABSTRACT

The neglect of buffering requirements in a classical job shop scheduling system often results in inapplicability in many complex real-world applications. To overcome this inapplicability, a new and more generalised scheduling problem is proposed under different stage-dependent buffering requirements and parallel use of identical-function machine units at each processing stage in job shop environments. The problem is formulated as a mixed integer programming model that can be exactly solved by ILOG-CPEX for small-size instances. Moreover, a hybrid metaheuristic algorithm embedded with a state-of-the-art constructive algorithm is developed. The computational experiment shows that the proposed metaheuristic can efficiently solve large-size instances. The result analysis indicates that the proposed approach can provide better configuration of real-world scheduling systems. The proposed DBPMJSS methodology has a potential to analyse, model and solve many industrial systems with the requirements of buffering conditions, particularly for manufacturing, railway, healthcare and mining industries.

1. Introduction

In the literature, the Job Shop Scheduling (**JSS**) problem with the makespan criterion is regarded as one of the most difficult problems in combinatorial optimisation. An indication of its difficulty is given by the fact that the famous 10-job 10-machine instance formulated for the first time by Muth and Thompson [1] was exactly solved by Carlier and Pinson [2], with a branch and bound algorithm that required about 17895 seconds (about 5 hours) of computing time on a PRIME 2655 computer. In the past 30 years, a considerable number of algorithmic improvements have been made and accumulated for the classical JSS problem. With regard to a literature review of JSS, Jain and Meeran [3] had provided a comprehensive overview of the progress, the techniques and the researchers involved in the classical JSS problem in 1990. Because of its significance from a theoretic view, the classical JSS problem is still an important research topic in the field of Operations Research in the last two decades. Due to its complexity, most of researchers recently focused on developing advanced metaheuristic algorithms to solve JSS in a more efficient way. For example, Zhang et al. [4] developed a Tabu Search (TS) metaheuristic with delicate neighbourhood structure and fast evaluation strategies for JSS. Furthermore, Zhang et al. [5] developed a hybrid metaheuristic by combining Simulated Annealing (SA) with TS to obtain high-quality JSS solution within reasonable CPU times. Udomsakdigool and Kachitvichyanukul [6] developed an Ant Colony Optimisation (ACO) algorithm with several specific features designed for JSS. Huang and Liao [7] developed a hybrid metaheuristic for JSS by combining ACO and TS mechanisms. Rego and Duarte [8] developed an efficient algorithm for JSS by combining the Shifting Bottleneck Procedure (SBP) algorithm with a Filter-&-Fan search procedure. Hasan et al. [9] developed a memetic algorithm (MA) with some priority rules for solving JSS. Lin et al. [10] developed a new Particle Swarm Optimisation (PSO) algorithm with a multi-type individual enhancement scheme for JSS. Yin et al. [11] developed a discrete Artificial Bee Colony (DABC) algorithm for JSS. Nasiri and Kianfar [12] developed an efficient TS metaheuristic with six types of neighbourhood moves for JSS. Peng et al. [13] presented an efficient TS metaheuristic with a path relinking procedure for JSS. Amirghasemi and Zamani [14] developed a fast genetic algorithm with the use of elite pool to obtain the optimal solutions of some well-known benchmark JSS instances within several seconds.

However, many realistic scheduling systems cannot be modelled as the classical JSS problem, due to the additional features such as the lack of storage units, zero-delaying-time restrictions, parallel use of machine units, and special service/technological requirements. In this case, the classical JSS problem should extended by considering these additional constraints. In recent years, mainly motivated by practical issues, research interest has arisen in the JSS problems with *Blocking* or *No-Wait*, called **NWJSS** or **BJSS** respectively in the literature. For example, the NWJSS problem

* Corresponding author.
E-mail addresses: sq.liu@qut.edu.au (S.Q. Liu), e.kozan@qut.edu.au (E. Kozan).

http://dx.doi.org/10.1016/j.cor.2016.04.023
0305-0548/

may arise in food industries, where the canning operation must immediately follow the cooking operation to ensure freshness. Another typical example is passenger train scheduling because express passenger trains should traverse continuously in certain track sections (e.g., tunnels) without any pauses [15–17]. In comparison, the BJSS problem permits a job either to leave immediately after processing if possible or to remain there until the downstream machine on the routing is available. The BJSS problem stems from a variety of industrial systems. For example, in a chemical factory, partially-processed chemical products sometimes must be kept in the processing machines because of temperature requirement or the non-existence of safe intermediary storage space. Another practical application is freight train timetabling that should seriously consider the blocking (wait-while-hold) constraints because a freight train sometimes has to remain on a track section until the next section on the routing becomes available [18]. Compared to the substantial body of knowledge in the classical JSS problem, the NWJSS and BJSS problems received surprisingly little attention from either a theoretical or computational perspective. An initial survey for this research area was given by Hall and Sriskandarajah [19]. Regarding the solution techniques, Mascis and Pacciarelli [20] studied the BJSS and NWJSS problems by means of alternative graph, which is an extension of the classical disjunctive graph. Furthermore, Meloni et al. [21] presented a rollout metaheuristic for the BJSS and NWJSS problems. The rollout metaheuristic corresponds to a constructive procedure that iteratively extends a partial schedule, represented by a partial selection of alternative arcs, to a complete schedule. Mati et al. [22] investigated a multi-resource BJSS problem, in which every operation simultaneously requires several resources such that deadlocks can be avoided. They extended a classical geometric approach to solve the two-job case. Brucker and Kampmeyer [23] proposed a tabu search (TS) metaheuristic algorithm for a cyclic BJSS problem. In their TS, the recovering procedures were developed to construct feasible neighbouring solutions in TS moves. Gröflin and Klinkert [24] devised a TS metaheuristic algorithm based on an extended graph model to solve BJSS efficiently. Samarghandi and ElMekkawy [25] developed a genetic algorithm to solve a two-machine NWJSS problem with a single sever and setup times. Pranzo and Pacciarelli [26] developed an iterated greedy algorithm to solve to two variants of BJSS with swap allowed and without swap allowed.

Another extension of the classical JSS problem is consideration of the flexibility of selecting alternative routes among machines in a job shop environment so that different processing times of an operation are optional in terms of machine (stage) assignment. In the literature, this problem type is called the Flexible Job Shop Scheduling (**FJSS**) problem and stated as follows: In an FJSS environment, it is assumed that there are a given set of jobs $J = \{J_1, J_2, \ldots J_j, \ldots J_n\}$ and a given set of machines $M = \{M_1, M_2, \ldots, M_m\}$. A job J_j is formed by a given sequence of operations $O_{1j} - O_{2j} - \ldots O_{ij} - \ldots - O_{oj}$. Each operation (O_{ij}) can be executed on any one machine from a specified subset of machines $M_{ij} \subseteq M$. Thus, the flexibility for operation O_{ij} exists if the number of machines in subset M_{ij} is multiple. However, such a flexibility is still limited as the specification of M_{ij} is operation-dependent. The idea of FJSS was first introduced by Brucker and Schlie [27] has become an entity of intensive research due to its usefulness in real-world production environment. Because of strong NP-hardness, most researchers were interested in applying metaheuristic algorithms to solve FJSS, as presented in the below. Brandimarte [28] proposed a two-level hierarchical approach for FJSS by tackling machine assignment and operation scheduling separately. Pezzella et al. [29] presented an efficient genetic algorithm with different selection and reproduction mechanisms for FJSS. Gao et al. [30] used the special chromosome structure and advanced operators to develop a hybrid genetic and variable neighbourhood descent algorithm for FJSS. Xing et al. [31] developed a Knowledge-Based ACO algorithm for FJSS by providing an effective integration between ACO and knowledge model. Bozejko et al. [32] developed a double-level parallel metaheuristic approach with machine selection module and operation scheduling module for solving FJSS. Zhang et al. [33] solved FJSS by developing an effective genetic algorithm based on an improved chromosome representation and diverse crossover and mutation operators. Yuan and Xu [34] developed a hybrid evolutionary-based memetic algorithm with harmony search and large neighbourhood search to solve large-scale FJSS instances. Türkyılmaz and Bulkan [35] developed a hybrid genetic algorithm with variable neighbourhood search for FJSS with the objective of minimising total tardiness. González et al. [36] developed a scatter search metaheuristic algorithm with effective neighbourhood structure to solve FJSS. Jia and Hu [37] solved a multi-objective FJSS problem by a path-relinking TS algorithm with the mechanism of back-jump tracking.

On the boundary of the job shop environment with No-Wait/Blocking, the JSS with Limited-Buffer (**LBJSS**) problem creates a relatively new direction of research but receives little attention [38]. However, the limited-buffer conditions often arise in most real-world production environments. For example, Toba [39] indicated from an application in semi-conductor industry that "Buffers accommodate lots (products) waiting at processing equipment units and supply the lots to the equipment units when they are ready to process them. The buffers alleviate abrupt throughput changes in production equipment by buffering several lots among contiguous process steps and prevent starvation of equipment units. As a result, the buffers have a major effect on the throughput performance of fabrication lines". Brucker and Kampmeyer [23] indicated that a feasible solution may be represented by the machine sequences of the jobs but the buffers should be incorporated in the solution representation. Brucker et al. [40] investigated the LBJSS problem by classifying the buffering requirements into the following three types.

i) job-dependent buffering: the buffering requirement of a job at a stage only depends on the job's characteristics;
ii) stage-dependent input buffering: a storage unit may store a job just before its processing at a stage; and
iii) stage-dependent output buffering: a storage unit may store a job just after its processing at a stage.

In this paper, the definition of the buffering requirement belongs to the above third type "*stage-dependent output buffering*". In addition to storage units, machine units at each stage in DBPMJSS have the identical operating function and are also stage-dependent. This is fundamentally distinct from FJSS in which a given subset of machines $M_{ij} \subseteq M$ (actually equivalent to stages in DBPMJSS) for one operation O_{ij} is operation-dependent. Moreover, the diffidence between FJSS and DBPMJSS is explained below by a benchmark FJSS instance called MK01 (see [28–30]).

1.1. MK01 FJSS data

10 6 2
6 2 ***1*** 5 ***3*** 4 3 ***5*** 3 ***3*** 5 ***2*** 1 2 ***3*** 4 ***6*** 2 3 ***6*** 5 ***2*** 6 ***1*** 1 1 ***3*** 1 3 ***6*** 6 ***3*** 6 ***4*** 3
5 1 2 6 1 3 1 1 1 2 2 2 6 4 6 3 6 5 2 6 1 1
5 1 2 6 2 3 4 6 2 3 6 5 2 6 1 1 3 3 4 2 6 6 6 2 1 1 5 5
5 3 6 5 2 6 1 1 1 2 6 1 3 1 3 5 3 3 5 2 1 2 3 4 6 2
6 3 5 3 3 5 2 1 3 6 5 2 6 1 1 1 2 6 2 1 5 3 4 2 2 6 4 6 3 3 4 2 6 6 6
6 2 3 4 6 2 1 1 2 3 3 4 2 6 6 6 1 2 6 3 6 5 2 6 1 1 2 1 3 4 2
5 1 6 1 2 1 3 4 2 3 3 4 2 6 6 6 3 2 6 5 1 1 6 1 3 1
5 2 3 4 6 2 3 3 4 2 6 6 6 3 6 5 2 6 1 1 1 2 6 2 2 6 4 6
6 1 6 1 2 1 1 5 5 3 6 6 3 6 4 3 1 1 2 3 3 4 2 6 6 6 2 2 6 4 6
6 2 3 4 6 2 3 3 4 2 6 6 6 3 5 3 3 5 2 1 1 6 1 2 2 6 4 6 2 1 3 4 2