Computers & Industrial Engineering 155 (2021) 107211

Contents lists available at ScienceDirect

Computers & Industrial Engineering

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

An effective MCTS-based algorithm for minimizing makespan in dynamic flexible job shop scheduling problem

Kexin Li [a], Qianwang Deng [a,*], Like Zhang [a], Qing Fan [a,b], Guiliang Gong [a], Sun Ding [a]

[a] State Key Laboratory of Advanced Design and Manufacturing for Vehicle Body, Hunan University, Changsha 410082, China
[b] State Key Laboratory of Construction Machinery, Zoomlion Heavy Industry Science and Technology Co., Ltd, Changsha 410013, China

ARTICLE INFO

Keywords:
Dynamic flexible job shop scheduling
Monte Carlo Tree Search
Rescheduling
Response time

ABSTRACT

In the past several decades, most of the research methods are designed to solve the static flexible job shop scheduling problem. However, in real production environments, some inevitable dynamic events such as new jobs arrival and machine breakdown may occur frequently. In this paper, we study a dynamic flexible job shop scheduling problem (DFJSP) considering four dynamic events, which are new jobs arrival, machine breakdown, jobs cancellation and change in the processing time of operations. A rescheduling method based on Monte Carlo Tree Search algorithm (MCTS) is designed to solve the proposed DFJSP with the objective of minimizing the makespan. Several optimization techniques such as Rapid Action Value Estimates heuristic and prior knowledge are adopted to enhance the performance of the MCTS-based rescheduling method. The response time to dynamic events is critical in DFJSP but has not been solved very well. To greatly reduce the response time to dynamic events, when dynamic events occur, multiple continuous specified time windows are designed for the proposed method, according to which the corresponding subsequent partial schedule for the remaining unprocessed operations is progressively generated. Some experiments have been conducted to compare the proposed method with the commonly used completely reactive scheduling methods and the GA-based rescheduling method. The experiment results indicate that the proposed method is an efficient and promising method for dynamic scheduling both on solution quality and computation efficiency.

1. Introduction

The Job shop scheduling problem (JSP) is a well-known strong NP-hard combinational optimization problem (Garey, Johnson, & Sethi, 1976), which mainly determines a schedule for a set of jobs with pre-specified operation sequences on the corresponding predefined machine to achieve some specific objectives. In practice, the widespread use of multipurpose machines allows the operation can be processed by multiple optional machines, which is a generalization of JSP and also known as the flexible job shop scheduling problem (FJSP). Hence, FJSP increases the flexibility of scheduling and is also considered to be strongly NP-hard.

Most of the literature on scheduling focuses on static scheduling problems and does not give much consideration to dynamic factors. However, in the actual manufacturing systems, the shop environment often changes dynamically due to some unpredictable dynamic real-time events, such as sudden machine breakdown or random new jobs arrival. In this case, the previously generated scheduling scheme may become less effective or even become infeasible. As stated in Ouelhadj and Petrovic (2009), dynamic scheduling is of great significance for the successful implementation of real manufacturing systems. Obviously, scheduling problems are dynamic in nature and more complicated than the static scheduling problems due to various random and uncertain events in reality. The FJSP turns into a new kind of problem which is called dynamic flexible job shop scheduling problem (DFJSP) when dynamic real-time events occur.

To solve the dynamic scheduling problem of manufacturing systems, there are mainly three categories of technologies, which are the completely reactive scheduling, the predictive-reactive scheduling and the robust pro-active scheduling (Ouelhadj & Petrovic, 2009). Among them, the predictive-reactive scheduling is the most commonly used method. This method has a scheduling or rescheduling process in which the previous scheduling is modified to accommodate the new shop environment caused by dynamic events. Most of the existing studies

* Corresponding author at: No. 2 South Lushan Road, Yuelu District Changsha, Hunan 410082, China.
E-mail addresses: lionelkx@hnu.edu.cn (K. Li), deng_arbeit@hnu.edu.cn (Q. Deng), likezhang@hnu.edu.cn (L. Zhang), gongguiliang@hnu.edu.cn (G. Gong), dings353849@hnu.edu.cn (S. Ding).

https://doi.org/10.1016/j.cie.2021.107211
Received 21 August 2020; Received in revised form 20 January 2021; Accepted 23 February 2021
Available online 3 March 2021

generated a new schedule by minimizing the impact of disruptions on shop production efficiency such as makespan (Adibi, Zandieh, & Amiri, 2010; Chryssolouris & Subramaniam, 2001; Liu, Fan, & Liu, 2015). In this case, it may obtain a new schedule that is completely different from the previous one, which is also called rescheduling.

Dispatching rules are often widely applied to solve dynamic scheduling problems in actual production systems due to their polynomial time complexity. Although dispatching rules can quickly respond to dynamic events, the solution quality is often not high enough to meet the requirements of efficient production. Therefore, many researchers have gradually used gene expression programming method to extract dispatching rules to solve the DFJSP under different specific constraints (Nie, Gao, Li, & Li, 2013; Nie, Gao, Li, & Shao, 2013; Ozturk, Bahadir, & Teymourifar, 2019; Teymourifar et al., 2020; Xu et al., 2020; Zhang, Mei, & Zhang, 2019). However, the development process of this method is still complicated, and the effectiveness of the extracted dispatching rules depends on the design of specialized genetic operators (Su et al., 2013).

Furthermore, against the shortcomings of dispatching rules, some meta-heuristic algorithms are widely used to solve the DFJSP. Rajabinasab and Mansour (2011) developed a multi-agent scheduling approach for a DFJSP considering dynamic events such as random jobs arrival, uncertain processing time and unexpected machine breakdown, and they compared their approach with five dispatching rules from literature. Gholami and Zandieh (2009) proposed an approach that integrates simulation into genetic algorithm to solve a DFJSP with random machine breakdowns. Reddy et al. (2018) studied a DFJSP that considers machine breakdown as a real-time event and proposed an effective hybrid evolutionary algorithm. Cao et al. (2019) proposed an adaptive scheduling algorithm inspired by the heterogeneous earliest finish time algorithm to address the makespan minimization in a DFJSP with new job arrival. Zadeh, Katebi, and Doniavi (2019) proposed a heuristic model inspired from artificial bee colony algorithm to address a DFJSP that minimizes makespan considering variable processing time of operations. Wang, Luo, and Cai (2017) developed an improved genetic algorithm based on variable interval rescheduling strategy to solve a DFJSP for minimizing makespan, which takes machine breakdown, urgent job arrival and job damage as disruptions into consideration. Zhang, Wang, and Liu (2017) proposed a two-layer scheduling method based on dynamic game theory, which was used as a real-time scheduling method to address a multi-objective DFJSP considering four kinds of dynamic events, and the objectives are to minimize makespan, total workload of machines and energy consumption. Shen and Yao (2015) studied a multi-objective DFJSP considering random new jobs arrival and machine breakdowns, and adopted a critical event-driven mode. They proposed a rescheduling method based on multi-objective evolutionary algorithm to regenerate new schedules.

Although there exist some research papers which study the DFJSP with several dynamic events such as machine breakdowns and new jobs arrival, there is still a great need to develop more efficient and effective methods. As pointed out by Kundakci and Kulak (2016), an obvious disadvantage of the traditional GA-based algorithm is that the computation time will increase rapidly with the increase of the total number of operations, because it contributes to the number of genes in chromosomes and reduces the efficiency of the GA operators. With the emergence of dynamic events, the previous schedule may become inefficient or even infeasible. Therefore, at this point, it needs to be repaired or a new rescheduling scheme needs to be generated. In the real manufacturing, the scale of the DFJSP is usually large and involves lots of operations to be assigned to an optional machine, which may lead to a significant deterioration in the computational efficiency of these meta-heuristic algorithms. In other words, it is a time-consuming process to generate a rescheduling scheme by meta-heuristic algorithms such as GA, especially when the scale of the problem is relatively large. However, most of the existing studies rarely take the computation time at each rescheduling point into account. In the actual production environment, too much computation time will lead to the delay of shop production and bring economic losses. Therefore, to meet the requirements for the practical production, it is necessary to develop a method that can quickly respond to dynamic events and generate a feasible and effective rescheduling scheme in an acceptable time.

Monte Carlo Tree Search (MCTS) is a best-first search algorithm, which finds optimal decisions in a given domain by taking random samples in the decision space and building a search tree according to the results (Browne et al., 2012). It has been widely and successfully used in the area of game playing, especially computer Go (Silver et al., 2016; Silver et al., 2017). It is especially useful in stochastic problems with very large or infinite state spaces. Some researchers have successfully applied MCTS algorithm in the field of scheduling (Asta et al., 2016; Furuoka & Matsumoto, 2017; Waledzik & Mandziuk, 2018). Wu, Wu, and Liang (2013) and Chou et al. (2015) have demonstrated the potential of using MCTS to solve the static FJSP. In the existing literature, MCTS has not yet been applied to solve scheduling problems under dynamic environments. As a complex combinational optimization problem, DFJSP has a huge state search space and it is a natural candidate problem for MCTS. Therefore, applying MCTS to solve DFJSP is a new idea and worth exploring in depth.

This paper studies a DFJSP considering four dynamic events, which are new jobs arrival, machine breakdown, jobs cancellation and change in the processing time of operations. In order to obtain an efficient solution in terms of solution quality and computational efficiency, we designed a MCTS-based rescheduling method. An event-driven mode is adopted, which means that rescheduling will be triggered once a dynamic event occurs, and the proposed method is executed immediately to generate a new rescheduling scheme. In order to improve the efficiency of MCTS, several optimization techniques are used in MCTS. Some well-known dispatching rules are employed to obtain the initial prior knowledge for MCTS. Considering that some traditional rescheduling methods require a large amount of time to generate a complete rescheduling scheme for all remaining unprocessed operations at each rescheduling point, and the MCTS can stop the tree generation at any time, multiple continuous specified time windows are designed for the MCTS-based rescheduling method. In this way, the MCTS-based rescheduling method can generate a partial rescheduling scheme corresponding to the subsequent specified time window at each rescheduling point. The proposed method can greatly reduce the response time to dynamic events since the number of operations in the partial schedule is relatively small. In addition, eighteen benchmarks of DFJSP are constructed based on some basic criteria to evaluate the performance of the proposed rescheduling method.

The main contribution of this paper can be summarized as follows: (1) A well-designed MCTS-based rescheduling method is proposed for the DFJSP. (2) Multiple continuous specified time windows are designed for the MCTS-based rescheduling method to progressively generate the partial schedule corresponding to the subsequent time window, which can effectively reduce the response time at each rescheduling point. (3) Eighteen benchmarks of DFJSP are constructed and the effectiveness of the proposed method is verified.

The remainder of this paper is organized as follows. In Section 2, the formal description of the DFJSP under consideration is outlined. In Section 3, A MCTS-based algorithm to achieve rescheduling is presented. In Section 4, the framework of the MCTS-based rescheduling method to solve DFJSP is given. The experimental design and results are discussed in Section 5. Finally, some conclusions and future works are given in Section 6.

2. Problem description

The DFJSP studied in this paper considers that n jobs are processed on m machines at the beginning of the schedule. During the actual production process, some dynamic events such as new job arrival and machine breakdown will inevitably occur. The DFJSP subjects to some

common assumptions in the following because of the high complexity.

(1) All machines are available at the beginning of the schedule.
(2) An operation of a job can be processed by only one machine at a time.
(3) Each machine can process at most one operation at a time.
(4) Once an operation has been processed on a machine, it must not be interrupted except for the machine breakdown. If an operation is interrupted by the machine breakdown, it must be processed from scratch on an optional available machine.
(5) An operation of a job cannot be processed until its previous operation is completed and there is no travel time between machines.
(6) The setup time of an operation is included in the processing time.
(7) Operations can wait to be processed in an unlimited buffer of a machine.

This study considers four common dynamic real-time events, which are change in the processing time of operations, new jobs arrival, machine breakdown and job cancellation. The details of dynamic events and their effects are discussed as follows.

(1) Change in the processing time of operations. Before the execution of the pre-scheduling scheme, the processing time of some operations may change due to the change of process technology. In this case, the scheduling scheme needs to be rebuilt with the new processing time of these operations. Therefore, the rescheduling is triggered when the processing time of operation is change.
(2) New jobs arrive dynamically over time. In real manufacturing system, new jobs may arrive continuously during the production process. In our DFJSP, the arrival time of new jobs is assumed to follow the Poisson distribution. Therefore, the time interval between job arrivals follows an Exponential distribution (Rangsaritratsamee, Ferrell, & Kurz, 2004; Sha & Liu, 2005; Vinod & Sridharan, 2008). The rescheduling is triggered when new jobs arrive, and according to the state of operations, the existing operations can be divided into four sets: the completed operations set, the operations set being processed, the unprocessed operations set and the new operations set. Obviously, all operations of new jobs are divided into the new operations set.
(3) Machine breakdown and repair. When a machine breaks down, the ongoing operation is interrupted and machine repair is carried out immediately. If the previous schedule is maintained, the start time and completion time of the unprocessed operations assigned to this machine will scheduled depending on the finishing time of the machine maintenance, and the makespan will increase when the maintenance time is long. So, the rescheduling is triggered when machine breakdown occurs. In our DFJSP, for each machine, the time interval between two consecutive machine breakdowns and the repair time for each maintenance are assumed to follow an Exponential distribution (Zandieh & Adibi, 2010).
(4) Job cancellation. In the production process, some customers may ask to cancel their orders. After jobs in these orders are cancelled, the remaining unprocessed operations of these jobs are removed from the unprocessed operations set and the rescheduling is triggered. In our DFJSP, the time of job cancellation is assumed to follow the Poisson distribution too. Therefore, the time interval between two adjacent job cancellations follows an Exponential distribution.

To evaluate the efficiency of the proposed rescheduling method, the makespan (i.e., the completion time of the last finished operation) is taken as the optimization objective, which is the most common criterion.

3. A MCTS-based algorithm to achieve rescheduling

3.1. Monte Carlo tree search

The MCTS has been the focus of much artificial intelligence research since it was first described. It has received considerable attention because of its great success and prolific achievements in the difficult problem of computer Go, and has proved beneficial in many other domains as well.

The MCTS is a best-first search algorithm, which finds the most promising moves in a given domain by selecting samples randomly in the search space and incrementally building a search tree in memory based on the results. The MCTS emphasizes the balance of exploration and exploitation, and it is an iterative method that performs the following four steps until reaching some predefined computational limitations such as time or iteration constraint. Here, we follow the constraint that executes a certain number of iterations.

(1) **Selection**: Starting from the root node, a child node selection policy is iteratively applied to descend through the tree until a nonterminal leaf node which has unexpanded child nodes is reached. This selection policy controls the tradeoff between exploitation and exploration. A widely used selection policy, the UCT (Upper Confidence bounds applied to Trees) (Kocsis & Szepesvari, 2006), is employed in this paper. This child node selection policy is to select the child node with the maximum UCT value. The specific formula is as follows.

$$UCT = X_j + C \times \frac{\sqrt{logN}}{N_j} \tag{1}$$

where X_j is the average value of the j-th child node, N is the visit count of the current (parent) node, N_j is the visit count of the j-th child node, and C greater than 0 is a constant which controls the tradeoff between exploitation and exploration.

(2) **Expansion**: When a leaf node is reached, one (or more) child nodes should be expanded from the leaf node and added to the tree.
(3) **Simulation**: Execute the simulation policy from the newly expanded node until a solution to the problem is generated, and an evaluation value of the final state is returned. In general, moves descending through the tree are selected either randomly or heuristically based on the selected simulation policy.
(4) **Backpropagation**: The evaluation value in the simulation is propagated through the path selected in the tree up to the root, and the node statistic information of these nodes are updated for later expansions.

An execution of these four phases is called an iteration. Details of our implementation for the MCTS-based algorithm to solve DFJSP will be discussed below.

3.2. Problem representations

When dynamic events occur, the job-shop status information is updated accordingly, and the previous scheduling scheme may become unavailable. After that, the DFJSP can be transformed into a static FJSP. Then, the proposed MCTS-based algorithm is used to generate a new rescheduling scheme. The scheduling contains two sub-problems: assigning operations to the corresponding available machines and sequencing the operations in each machine. The following describes how to map a FJSP to a tree topology structure.

Assume that a tree topology structure with a root node R represents the initial state, and each node extended to this tree is corresponded to a candidate operation-machine pair information. Each node except the

Table 1
A FJSP example with 2 jobs and 4 machines.

Jobs	Operations	Optional Machines and corresponding processing time			
Job1	O_{11}	M_1 (2)	M_2 (6)	M_3 (5)	M_4 (3)
	O_{12}	—	M_2 (8)	—	M_4 (4)
	O_{21}	M_1 (3)	—	M_3 (6)	—
Job2	O_{22}	M_1 (4)	M_2 (6)	M_3 (5)	—
	O_{23}	—	M_2 (7)	—	M_4 (5)

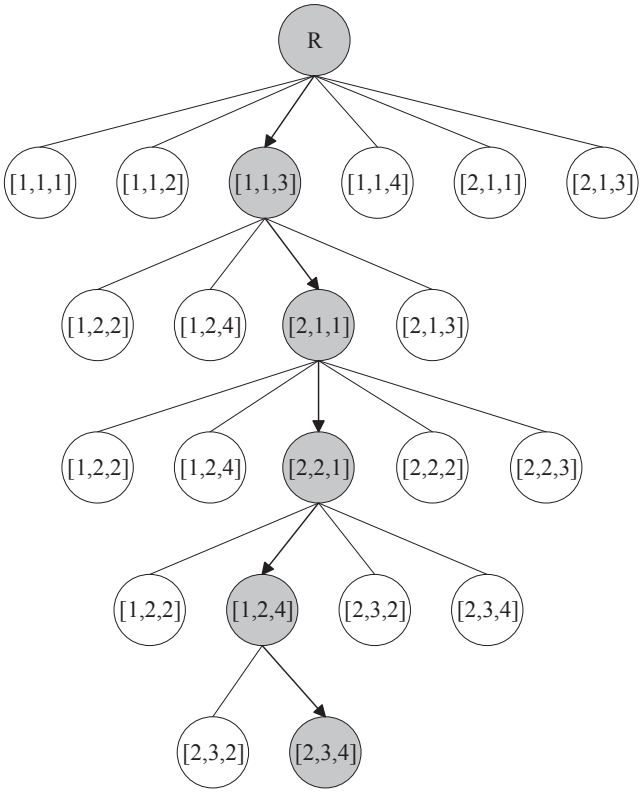

Fig. 1. The construction of tree topology structure for the 2 × 4 FJSP example.

root node R has a unique parent node and each non-leaf node has some legal child nodes. So, the location information of the nodes in the tree represents the sequence information of the operations.

For the purposes of discussion, we use the following notations. Let N_{op} denote the total number of operations. Let $[i, j, k]$ denote the operation-machine pair information which is matched with the node in the tree, and it denotes the jth operation of job i is processed on machine k. Note that $[i, j, k]$ is called a move in this study.

For a detailed description of this mapping transformation, a 2 × 4 FJSP example is listed in Table 1. The construction of tree topology structure corresponding to this example is shown in Fig. 1. The information on each child node in the figure is legal operation-machine pairs for the operations waiting to be processed.

The mapping construction process is described as follows:

Step 1: Set the node R as the root node and initialize an empty schedule S.
Step 2: Repeat the four steps of MCTS for a certain number of times which is denoted as N_s.
Step 3: After the previous step is completed, select the most visited child node of the root node and add the move matched with it to the schedule S. Meantime, set this selected child node as the new root node. For example, when the current root node is R, after N_s MCTS iterations, the node associated with the move [1, 1, 3] is selected. Then this move is added to the schedule S, and the node is set to the new root node.
Step 4: Repeat Step 2–3 N_{op} times, and all operations are scheduled. Then, a complete schedule is obtained. Take Fig. 1 for example, the initial root node R and all selected nodes are marked with gray shadow. Starting from the initial root node R, the corresponding moves of these nodes form a complete schedule.

3.3. Decoding

A schedule solution can be decoded into semi-active, active, non-delay and hybrid schedules. The active schedule is suitable to be adopted because the makespan is a regular criterion (Li & Gao, 2016). Therefore, this decoding method is adopted in this paper. That is, an operation is inserted into the first available idle time interval of its assigned machine as early as possible.

3.4. Some optimization techniques used to improve MCTS

Since the search space of FJSP is too large, in order to improve the efficiency of MCTS, we decided to adopt the following optimization techniques.

3.4.1. Subtree keeping policy

After a certain number of MCTS iterations, the best child node of the root node is selected as the new root node. Instead of building the subtree of the selected node from scratch, the existing subtree of this new root node in the previous tree is retained for the subsequent iterations. In this way, the information obtained from all previous iterations of each node in the retained subtree, including the evaluation value and the number of visits, will be reused. This policy can obtain a more accurate evaluation value of each node in the retained subtree and expand the search depth and width of MCTS.

3.4.2. Rapid action value estimation

Rapid Action Value Estimation (RAVE) is a popular enhancement of the All-Moves-As-First (AMAF) heuristic. Gelly and Silver (2007) first attempted to combine AMAF with UCT in the field of computer Go, and based on this they proposed the RAVE heuristic in Gelly and Silver (2011) later.

The RAVE uses the AMAF to estimate the value of each move from each node in the search tree. The RAVE provides an easy way to share knowledge of relevant nodes in the search tree to quickly get a biased estimate value of the move. This biased estimate is often used to determine the best move after only a few iterations, which can significantly improve the performance of the MCTS algorithm. Therefore, we consider adding RAVE to MCTS when doing selection. The final evaluation value is a weighted sum of the UCT value and the RAVE value, and the formula is as follows.

$$Value_j = (1-\beta) \times UCT_j + \beta \times RAVE_j \tag{2}$$

where β is a weight parameter used to balance the UCT value and RAVE value.

Similar to the UCT algorithm, an exploration reward is added to maintain the balance between exploration and exploitation, and the final formula is as follows.

$$Value_j^* = Value_j + C \times \sqrt{\frac{logN}{N_j}} \tag{3}$$

3.4.3. Prior knowledge

When expanding a node, if we do not initially get some useful information (i.e., prior knowledge) about the next move, we can only set the same initial value for each newly expanded node. In order to make full use of the results of previous iterations, we record these results into a

global array K_{prior}. The size of K_{prior} is $N_{op} \times m \times N_{op}$, which is indexed by the modified move ($[i, j]$, k, seq). It represents that the jth operation of job i is assigned on the seqth position of machine k. The array K_{prior} records the average estimated value and the visit count of moves obtained so far, and they will be updated after each iteration. Then, find the next move with the maximum estimated value in K_{prior}, and the child node associated with it will be expanded in the current expansion phase of MCTS.

The array K_{prior} is built at the beginning of the algorithm, and both the initial value and the visit count of all possible moves are set to 0. Once a complete schedule S and its evaluation value v are obtained during the simulation phase of each MCTS iteration, the estimated value and the visit count of all moves included in this schedule are updated to the array K_{prior} with the following steps.

Step 1: Let d be the first move in schedule S.

Step 2: Update the estimated value and the visit count of $K_{prior}(d)$, respectively.

(1) Update the estimated value.

$$value = \frac{value \times visit + v}{visit + 1} \tag{4}$$

(2) Update the visit count.

$$visit = visit + 1 \tag{5}$$

Step 3: If there is no operation in S, stop; else, let d be the next move in S and go to Step 2.

Through the above steps, we can get the initial value of the newly expanded node based on the array K_{prior}.

3.4.4. Initialized prior knowledge

Initialized prior knowledge can provide useful information about whether each legal move is good or bad during the expansion phase of MCTS. Based on Section 3.4.3, in order to reduce the initialization time as much as possible, some commonly used completely reactive scheduling methods are adopted to initialize the initial value and the visit count of the array K_{prior}. The completely reactive scheduling methods assign operations to their optional machines according to a specific machine assignment rule, and once a machine becomes idle, select the operation with the highest priority based on a heuristic priority dispatching rule when there are operations in the waiting queue (Shen & Yao, 2015).

Three machine assignment rules (MARs) for all operations are employed in this paper. The first rule is to assign each operation to its available machine with the shortest processing time. The second rule is to find the available machine with the minimum workload currently for each operation waiting to be processed, then assign the operation to that machine and updates the workload of that machine. The third rule is to assign each operation to its available machine at random. In brief, this paper refers to them as MAR1, MAR2 and MAR3, respectively. In addition, three common priority dispatching rules (PDRs) are used in this section, which includes Shortest Processing Time First (SPT), First In First Out (FIFO) and Last In First Out (LIFO). The details of Initialized Prior Knowledge are as follows.

Firstly, a population with one thousand chromosomes is initialized, 90% of which are generated with a 10% mixture of nine combination methods of three MARs and three PDRs mentioned above, then the rest is randomly generated. Secondly, each chromosome is decoded to get a complete schedule and obtain its estimated value. Thirdly, the estimated value and the visit count of all moves included in this schedule are updated to the array K_{prior} for initialization.

3.5. Our modified MCTS

In our modified MCTS, the final evaluation value Z can be calculated by the following evaluation function:

$$Z = 2 - \frac{C_{max}}{best(C_{max})} \tag{6}$$

where $C_{\max}$ is the makespan of the current schedule, and $best(C_{\max})$ is the minimum makespan found so far.

Based on the proposed optimization techniques, the four phases of MCTS are modified as follows:

(1) **Selection**: Starting from the root node, select the child node with the maximum *Value** recursively according to the formulas (2) and (3) until a nonterminal leaf node is reached. The values of parameter β and C are all set to 0.5 in this paper.
(2) **Expansion**: When a leaf node is reached, a child node with the maximum prior value will be expanded from the leaf node and added to the tree.
(3) **Simulation**: Perform the simulation policy recursively from the newly expanded node until all operations are scheduled. The simulation policy can be random simulation or choose the move with the maximum prior value. In order to improve the simulation quality and increase the diversity, we decided to use a probabilistic selection strategy. That is, choose the move randomly with probability p, and choose the move with the maximum prior value with probability $1 - p$. We set p to 0.4 in this paper.
(4) **Backpropagation**: The evaluation value is calculated based on the simulation result, then the evaluation value and visit count of nodes are propagated and updated through the path selected in the tree up to the root. This backpropagation includes to update the evaluation value and visit count of RAVE of relevant nodes.

4. Framework of the MCTS-based rescheduling method to solve DFJSP

4.1. Rescheduling mode

In the actual production, many dynamic uncertain events may occur frequently. When a dynamic event occurs, the previous scheduling scheme may become less effective and even unavailable. Rescheduling is required to adapt to the dynamic changes in the production environment.

Dynamic scheduling can be divided into completely reactive scheduling, predictive-reactive scheduling and robust pro-active scheduling (Mehta & Uzsoy, 1999; Vieira, Herrmann, & Lin, 2003). This paper studies the predictive-reactive scheduling method for the DFJSP. This method can be classified as either continuous rescheduling which reschedule the unprocessed operations each time a dynamic event such as machine breakdown occurs, or periodic rescheduling which rescheduling after a fixed time interval (Church & Uzsoy, 1992). In this paper, the continuous rescheduling method which is also referred to event-driven rescheduling is adopted, because it has a quick response to dynamic events and can achieve better scheduling performance. In other words, rescheduling is performed once a dynamic event occurs. The time at which the rescheduling method is triggered is called the rescheduling point.

4.2. The flowchart of the modified MCTS-based method for DFJSP

In most real manufacturing systems, the shop status information usually needs to be updated in real time. Rescheduling is required when dynamic events occur. During the execution of the schedule, at each rescheduling point, the MCTS-based rescheduling method is triggered to

generate a new schedule and the operation sequence on each machine is reassigned. The newly generated schedule is executed in the job shop until the next rescheduling point comes, at which time the rescheduling method is triggered again.

To solve DFJSP, at each rescheduling point, some traditional methods generate a complete schedule for all remaining unprocessed operations and then execute this schedule in the job shop. Different from these methods, the MCTS-based algorithm can stop the tree generation at any time and get a partial schedule. This is an inspiration for us to reduce the calculated response time at each rescheduling point. To be specific, the MCTS-based algorithm can generate a partial schedule corresponding to the subsequent specified time window on the basis of global consideration of all remaining unprocessed operations, rather than a complete schedule for all unprocessed operations. This is equivalent to dividing all unprocessed operations according to a pre-fixed time window, and then scheduling these corresponding partial operations. Therefore, we designed multiple continuous specified time windows for the MCTS-based algorithm to limit the scheduling horizon of the subsequent partial schedule for the remaining unprocessed operations. The details of the entire scheduling process are as follows.

In our proposed rescheduling method, a rolling horizon is considered. While executing the partial schedule of the current time window in the job shop, a partial schedule of the next time window is generated simultaneously, and the same is true when executing the partial schedule of the next time window. For the sake of description, we call this step the normal step here. If a dynamic event occurs in the current time window, the partial schedule of the next time window that has been generated in advance will be abandoned and the rescheduling is triggered. In this case, there is a response time to the dynamic event. After generating a new partial rescheduling scheme, then repeat the normal step until another dynamic event occurs. But if no dynamic event occurs in the current time window, the partial schedule of the next time window that has been generated in advance will be executed normally, and then repeat the normal step until another dynamic event occurs.

To be specific, suppose that the length of the time window is fixed as w. The partial schedule within the time window $[0, w]$ is generated by the MCTS-based algorithm at time 0. Following the normal step, during the execution of this schedule in $[0, w]$, the partial schedule of the subsequent time window $[w, 2w]$ is also generated by the MCTS-based algorithm in advance. Then the normal step is repeated until a dynamic event occurs at time T_1. Assume that $kw < T_1 \leq (k+1)w$ $(k = 0, 1, \ldots)$. Obviously, these partial schedules within the time window $[0, kw]$ are executed one after another. The partial schedule within the time window $[kw, T_1]$ is executed, but the partial schedules within the time window $[T_1, (k+1)w]$ and $[(k+1)w, (k+2)w]$ will not be executed, because the rescheduling is triggered at time T_1. Assume that the response time at each rescheduling point is ignored, the partial schedule S_1 of the subsequent time window $[T_1, T_1 + w]$ is generated by the MCTS-based algorithm at time T_1. During the execution of schedule S_1 in $[T_1, T_1 + w]$, following the normal step, the partial schedule S_2 of the subsequent time window $[T_1 + w, T_1 + 2w]$ is also generated by the MCTS-based algorithm in advance. This continues until the next dynamic event occurs and the rescheduling is triggered again. Then the above process is repeated until all operations are processed.

The proposed MCTS-based rescheduling method for DFJSP can be summarized as follows: (1) During the execution of a partial schedule of the current time window, a partial schedule of the next time window has been generated by the modified MCTS algorithm in advance. (2) When the rescheduling is triggered, update the job-shop status information which includes the available time of machines, the finished operation set, the unprocessed operation set and so on. In this case, the subsequent partial schedule generated in the current time window will not be executed. (3) At the rescheduling point, perform the modified MCTS algorithm to generate a partial schedule of the subsequent time window. (4) Repeat the above three steps until all operations are processed. The flowchart of the modified MCTS-based method for DFJSP is summarized in Fig. 2.

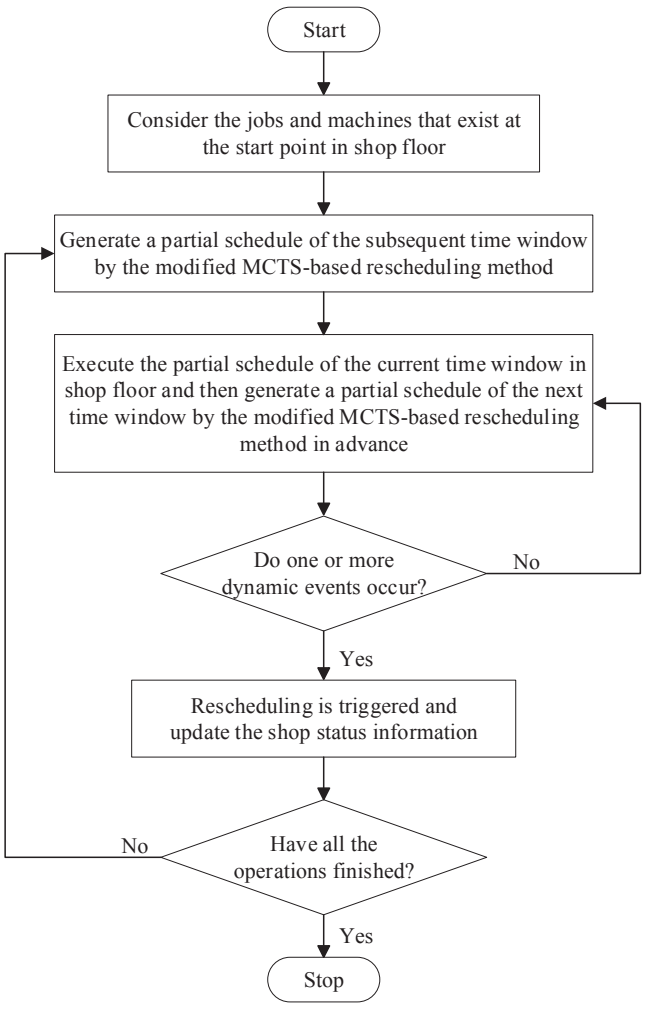

Fig. 2. Flowchart of the modified MCTS-based method for DFJSP.

5. Experimental studies and discussion

In this work, we conduct two sets of experiments. The first set of experiments is meant to investigate the effectiveness of the optimization techniques used to improve MCTS. The second set of experiments is implemented to evaluate the performance of the proposed MCTS-based method for DFJSP by comparing it with some existing rescheduling methods. The proposed MCTS-based method is coded by Python programming language and implemented on a PC configured with an Intel (R) Core (TM) i7-8750H CPU of 2.2 GHz and 8 GB RAM.

5.1. Experiment 1

This group of the experiments is carried out to examine the performance of the techniques used to improve MCTS which mentioned in Section 3.4. The performance is tested by the benchmark problem instances MK01-MK10 from Brandimarte (1993), which is well-known in the static flexible job shop scheduling problem. This problem set consists of 10 benchmark instances in which the total numbers of operations vary from 55 to 240.

In this subsection, we compare five versions of MCTS-based algorithm by incrementally adding Subtree Keeping Policy, RAVE, Prior Knowledge and Initialized Prior Knowledge, which are named as MA1, MA2, MA3, MA4 and MA5, respectively. In other words, the next version has one more optimization technique than the previous one. To be

Table 2
Comparisons of five versions of MCTS-based algorithms.

Problem	$n \times m$	MA1		MA2		MA3		MA4		MA5	
		ARPD	CPU	ARPD	CPU	ARPD	CPU	ARPD	CPU	ARPD	CPU
MK01	10 × 6	5.40	12.7	4.40	12.6	2.95	18.2	1.98	18.0	**0.00**	19.8
MK02	10 × 6	13.39	16.2	11.51	15.9	8.29	20.7	4.48	22.2	**0.00**	22.4
MK03	15 × 8	0.59	107.8	0.29	112.5	**0.00**	151.3	**0.00**	162.7	**0.00**	164.7
MK04	15 × 8	9.97	33.2	7.66	37.0	6.19	50.5	5.01	51.7	**0.29**	54.4
MK05	15 × 4	4.45	50.7	3.80	53.8	2.17	81.4	1.30	87.5	**0.11**	87.7
MK06	10 × 15	18.88	108.0	15.19	112.5	6.60	145.5	2.87	156.0	**1.08**	177.3
MK07	20 × 5	12.77	49.6	10.48	51.5	6.52	70.5	4.48	74.9	**0.00**	83.4
MK08	20 × 10	1.61	239.4	1.03	268.5	0.84	410.5	0.31	410.4	**0.08**	431.1
MK09	20 × 10	7.49	280.0	6.56	310.6	2.73	466.9	1.10	471.7	**0.00**	492.9
MK10	20 × 15	10.54	290.2	9.34	321.3	3.47	481.3	0.98	486.6	**0.00**	504.0

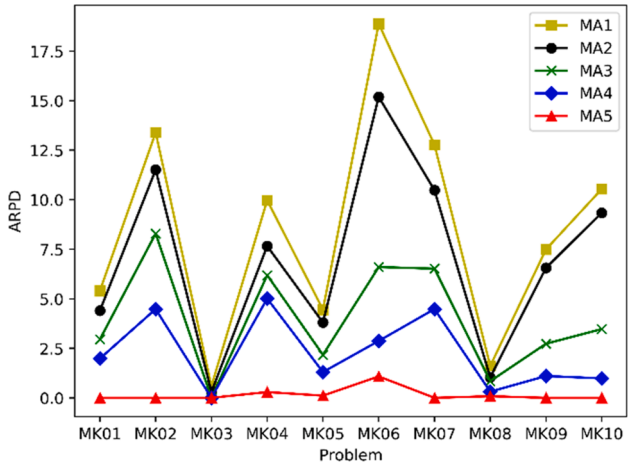

Fig. 3. The ARPD value of the five algorithms.

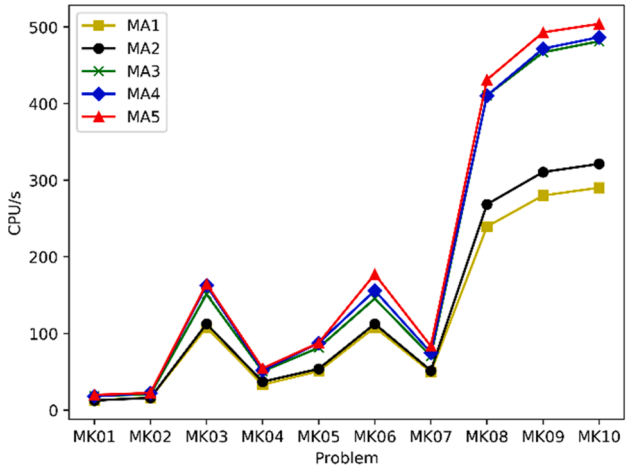

Fig. 4. The CPU time of the five algorithms.

specific, MA1 does not include any optimization techniques, MA2 includes Subtree Keeping Policy only, MA3 includes Subtree Keeping Policy and RAVE, and so on.

For a given instance, we calculate the relative percent deviation (RPD) and the average RPD (ARPD) (Zhou et al., 2018), and they are expressed as follows.

$$RPD = \frac{Alg_i - Min_i}{Min_i} \times 100 \tag{7}$$

where Alg_i is the result obtained by a given algorithm in the ith run and Min_i is the minimum result among all the results obtained by all algorithms in the ith run.

$$ARPD = \frac{\sum_{i=1}^{l} RPD}{l} \tag{8}$$

where l is the number of runs for each instance.

Every instance was run 5 times independently. For each instance, the ARPD is used to measure the results obtained by these five algorithms. Obviously, the smaller the ARPD value is, the better the performance of the corresponding algorithm is.

The number of iterations for each MCTS-based algorithm is set to 200. Table 2 shows the comparison results of the five algorithms. In this table, $n \times m$ means that this problem contains n jobs and m machines, the CPU represents the mean computational time (s) of five runs, and the best ARPD value for each instance is highlighted in bold.

The ARPD value and the CPU time of the five algorithms for each instance are plotted in Figs. 3 and 4, respectively. Although we can see that the CPU time increases as the number of optimization techniques is added, the performance of corresponding MCTS-based algorithm has been improved significantly for each instance. Therefore, we decided to apply algorithm MA5, which combines all the proposed optimization techniques, to the subsequent experiment.

5.2. Experiment 2

It has been indicated that a job shop with more than six machines can illustrate the complexity and difficulty of the large dynamic job shop scheduling problem (Adibi, Zandieh, & Amiri, 2010; Chryssolouris & Subramaniam, 2001). In this paper, a flexible job shop with eight machines ($m = 8$) is simulated to evaluate the performance of the proposed method.

There is no benchmark instance for DFJSP so far. This paper designed dynamic flexible job shop problem instances based on a static flexible job shop benchmark problem instances MK04 from Brandimarte (1993). These instances consider four dynamic events which are new jobs arrival, machine breakdown, change in the processing time of operations and job cancellation.

The number of operations for each new job is uniformly distributed over the interval of $[1, m + 2]$. The number of optional machines for each new operation is selected at random from the set $\{1, 2, \ldots, m\}$. The processing time of each new operation follows the uniform distribution on the interval of [1, 10]. The changed processing time of an operation also varies uniformly within [1, 10]. For each machine, the time interval between failures (TBF) and the time to repair (TTR) are assumed to follow an exponential distribution. To make the simulation more realistic, the mean time between failures (MTBF) and the mean time to repair (MTTR) are different for each machine. The MTBF and MTTR are uniformly distributed over [50, 70] and [10, 20], respectively. These values are selected so that the average available time of a machine is 60 units of time, and then the average time to repair a failure is 15 units of time. Thus, the availability of a machine is 80%. Eighteen different

Table 3
The GA parameters.

Parameters	
The size of the population, *Popsize*	400
The total number of generations, *maxGen*	200
The permitted maximum step size with no improving, *maxStagnantStep*	20
Reproduction probability, *Pr*	0.005
Crossover probability, *Pc*	0.8
Mutation probability, *Pm*	0.1

number of new jobs arrival are set, which are 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 120, 140, 160, 180, 200, 220, 240 and 260. Specifically, each number represents the total number of new jobs arrived for each DFJSP instance. The time interval between job arrivals (TBJA) and the time interval between job cancellations (TBJC) are assumed to follow an exponential distribution. The mean time between job arrivals (MTBJA) and the mean time between job cancellations (MTBJC) are set to 20 and 60, respectively. The number of new jobs per arrival is assumed to be 5, and the number of jobs per cancellation is 1.

All the designed DFJSP instances are based on a 15×8 static flexible job shop problem, where the initial number of jobs and machines are 15 and 8, respectively. These DFJSP instances are generated based on the parameters described above, which are provided in Appendix A. Since there are 18 different total number of new jobs arrival, total of 18 instances named D01-D18 are conducted. Instances which have less than 60 jobs (D01-D05) are categorized as small size instances. Instances which have 60 to 100 jobs (D06-D10) are categorized as medium size instances, and instances which have more than 100 jobs (D11-D18) are categorized as large size instances. For each instance, the simulation continues until all operations including all operations of new jobs are completed. All dynamic events that occur during the scheduling horizon are considered, and other parameters in the dynamic environment are the same in each experiment.

In order to further verify the effectiveness and efficiency of the proposed MCTS-based rescheduling method considering some dynamic events in the real-time flexible job-shop environment, we compared it with the commonly used completely reactive scheduling methods. In addition, based on the event-driven rescheduling mode, we compared it with GA proposed in Li and Gao (2016), and the GA parameters we set are the same as it, which are shown in Table 3. Specifically, when rescheduling is triggered, the GA is executed to generate a new schedule, and we call it GA-based rescheduling method here.

For the completely reactive scheduling methods, three machine assignment rules called MAR1, MAR2 and MAR3 are employed in this paper. Four common priority dispatching rules are adopted, which includes SPT, FIFO, LIFO and Random. When rescheduling is triggered, these unprocessed operations need to be assigned to an available machine. For example, the operations of these newly arrived jobs, the operations in the waiting queue of the broken machines and some operations that previously cannot be processed due to the breakdown of available machines become processable again because of the machine repairs.

Based on the designed DFJSP instances, twelve Combination methods of three MARs and four PDRs, the MCTS-based rescheduling method and the GA-based rescheduling method were evaluated and compared in this paper. The number of iterations for MCTS algorithm is set to 200, and the length of the specified time window of the MCTS-based rescheduling method is set to 5. Each instance was run 10 times independently.

5.2.1. *The experimental results*

The makespan comparisons of these rescheduling methods in small and medium size instances and in large size instances are listed in Table 4 and Table 5, respectively. The ARPD value comparisons of these rescheduling methods in small and medium size instances and in large size instances are listed in Table 6 and Table 7, respectively. In addition,

Table 4
The average makespan comparisons of the MCTS-based method against the traditional rescheduling methods in small and medium size instances.

Instances	D01	D02	D03	D04	D05	D06	D07	D08	D09	D10
Number of new jobs	10	20	30	40	50	60	70	80	90	100
$n \times m$	25×8	35×8	45×8	55×8	65×8	75×8	85×8	95×8	105×8	115×8
MAR1 + SPT	236.0	279.0	298.0	324.2	379.3	422.8	441.9	483.4	492.0	513.2
MAR1 + FIFO	226.4	281.8	301.3	328.2	346.2	416.7	433.0	473.2	479.3	502.8
MAR1 + LIFO	230.4	281.1	298.7	328.8	359.9	417.8	432.9	472.6	479.1	503.6
MAR2 + SPT	111.4	137.8	173.3	227.4	296.5	326.1	368.2	430.3	445.2	483.5
MAR2 + FIFO	110.4	140.1	172.7	226.7	297.2	326.0	366.4	431.2	447.1	483.8
MAR2 + LIFO	113.2	138.9	172.5	225.8	299.9	327.0	366.3	433.1	445.5	485.1
MAR3 + SPT	129.1	184.5	216.2	277.2	325.0	375.1	426.1	503.5	527.9	558.2
MAR3 + FIFO	130.3	186.8	222.6	281.2	331.9	378.4	428.8	515.1	534.5	563.2
MAR3 + LIFO	128.3	184.8	222.2	282.1	327.1	376.2	434.4	513.6	534.8	567.1
GA-based rescheduling method	85.2	125.1	153.4	197.2	279.0	306.7	336.8	409.5	416.6	458.0
MCTS-based rescheduling method	87.2	125.8	159.3	202.5	279.5	312.3	341.9	415.4	425.9	465.3

Table 5
The average makespan comparisons of the MCTS-based method against the traditional rescheduling methods in large size instances.

Instances	D11	D12	D13	D14	D15	D16	D17	D18
Number of new jobs	120	140	160	180	200	220	240	260
$n \times m$	135×8	155×8	175×8	195×8	215×8	235×8	255×8	275×8
MAR1 + SPT	553.8	589.3	693.0	705.4	889.0	989.0	1094.1	1246.0
MAR1 + FIFO	553.3	602.9	693.0	714.7	889.0	989.0	1102.4	1246.5
MAR1 + LIFO	550.6	603.1	693.0	709.5	889.0	989.0	1103.1	1242.9
MAR2 + SPT	550.0	604.7	683.0	739.4	869.1	938.9	1052.5	1224.2
MAR2 + FIFO	546.9	608.4	684.4	737.8	868.4	945.6	1053.8	1233.0
MAR2 + LIFO	551.2	609.9	685.5	739.6	869.2	945.1	1056.8	1222.9
MAR3 + SPT	612.1	686.1	784.8	846.6	939.2	1063.9	1171.4	1311.4
MAR3 + FIFO	629.9	704.9	814.3	875.6	972.2	1120.6	1223.3	1410.4
MAR3 + LIFO	631.6	708.4	809.5	878.7	976.0	1125.0	1229.8	1406.9
GA-based rescheduling method	495.8	553.3	619.6	676.6	801.2	870.8	985.1	1082.5
MCTS-based rescheduling method	518.1	564.7	640.6	696.6	833.6	909.7	1012.5	1129.1

Table 6
The ARPD value comparisons of the MCTS-based method against the traditional rescheduling methods in small and medium size instances.

Instances	D01	D02	D03	D04	D05	D06	D07	D08	D09	D10
Number of new jobs	10	20	30	40	50	60	70	80	90	100
$n \times m$	25 × 8	35 × 8	45 × 8	55 × 8	65 × 8	75 × 8	85 × 8	95 × 8	105 × 8	115 × 8
MAR1 + SPT	177.65	124.28	94.45	64.42	35.95	37.91	31.40	18.06	18.09	12.19
MAR1 + FIFO	166.35	126.54	96.60	66.44	24.09	35.90	28.76	15.57	15.05	9.92
MAR1 + LIFO	171.06	125.97	94.93	66.75	29.00	36.28	28.73	15.43	15.00	10.10
MAR2 + SPT	31.06	10.78	13.06	15.32	6.27	6.37	9.48	5.10	6.87	5.70
MAR2 + FIFO	29.88	12.63	12.68	14.96	6.52	6.33	8.95	5.31	7.32	5.74
MAR2 + LIFO	33.18	11.66	12.59	14.51	7.49	6.66	8.93	5.79	6.94	6.03
MAR3 + SPT	51.88	48.30	41.03	40.56	16.49	22.35	26.70	22.97	26.72	22.04
MAR3 + FIFO	53.29	50.17	45.20	42.62	18.96	23.43	27.51	25.82	28.30	23.12
MAR3 + LIFO	50.94	48.56	44.98	43.05	17.24	22.70	29.18	25.44	28.38	24.00
GA-based rescheduling method	0.24	0.56	0.00	0.00	0.00	0.03	0.15	0.00	0.00	0.09
MCTS-based rescheduling method	2.59	1.13	3.93	2.69	0.18	1.86	1.67	1.46	2.23	1.72

Table 7
The ARPD value comparisons of the MCTS-based method against the traditional rescheduling methods in large size instances.

Instances	D11	D12	D13	D14	D15	D16	D17	D18
Number of new jobs	120	140	160	180	200	220	240	260
$n \times m$	135 × 8	155 × 8	175 × 8	195 × 8	215 × 8	235 × 8	255 × 8	275 × 8
MAR1 + SPT	11.70	6.51	11.86	4.25	11.01	13.59	11.72	15.11
MAR1 + FIFO	11.60	8.98	11.86	5.64	11.01	13.59	12.56	15.16
MAR1 + LIFO	11.06	9.01	11.86	4.87	11.01	13.59	12.65	14.82
MAR2 + SPT	10.93	9.28	10.24	9.29	8.52	7.84	7.43	13.10
MAR2 + FIFO	10.31	9.96	10.47	9.05	8.44	8.61	7.61	13.91
MAR2 + LIFO	11.18	10.23	10.64	9.32	8.53	8.55	7.92	12.99
MAR3 + SPT	23.45	24.00	26.66	25.13	17.27	22.19	19.60	21.16
MAR3 + FIFO	27.05	27.41	31.42	29.42	21.40	28.68	24.92	30.30
MAR3 + LIFO	27.39	28.03	30.68	29.88	21.88	29.20	25.59	29.97
GA-based rescheduling method	0.00	0.00	0.00	0.00	0.00	0.00	0.49	0.00
MCTS-based rescheduling method	4.50	2.06	3.40	2.96	4.09	4.48	3.37	4.31

the total computation time (s) (which called TCPU) of MCTS-based rescheduling method and the TCPU time of GA-based rescheduling method in each run are recorded, both of which represent the sum of the time they take each time a subsequent schedule is generated. Moreover, the computation time of generating a new rescheduling scheme at each rescheduling point is taken as the response time at that rescheduling point, and the total response time (s) of all rescheduling point (which called TR) of MCTS-based rescheduling method and the TR time of GA-based rescheduling method in each run are also recorded. The mean TCPU time and the mean TR time are taken as the final results, and the comparisons of these two rescheduling methods in small and medium size instances and in large size instances are listed in Table 8 and Table 9, respectively. In these tables, $n \times m$ means that this instance contains n jobs and m machines.

Table 8
Comparisons of the mean TCPU time and the mean TR time of MCTS-based rescheduling method and GA-based rescheduling method in small and medium size instances.

Instances	D01	D02	D03	D04	D05	D06	D07	D08	D09	D10
Number of new jobs	10	20	30	40	50	60	70	80	90	100
$n \times m$	25 × 8	35 × 8	45 × 8	55 × 8	65 × 8	75 × 8	85 × 8	95 × 8	105 × 8	115 × 8
GA-based method	455.5 (455.5)	552.1 (552.1)	643.7 (643.7)	876.6 (876.6)	979.9 (979.9)	1146.3 (1146.3)	1395.8 (1395.8)	1973.6 (1973.6)	2438.8 (2438.8)	3388.6 (3388.6)
MCTS-based method	184.5 (103.4)	272.8 (167.6)	339.4 (201.2)	579.3 (355.9)	772.2 (470.8)	927.9 (537.8)	1131.5 (628.4)	1616.4 (937.2)	1829.8 (1043.8)	2187.9 (1310.7)

The values $x(y)$ in this table denote that the mean TCPU time and the mean TR time are x and y, respectively.

Table 9
Comparisons of the mean TCPU time and the mean TR time of MCTS-based rescheduling method and GA-based rescheduling method in large size instances.

Instances	D11	D12	D13	D14	D15	D16	D17	D18
Number of new jobs	120	140	160	180	200	220	240	260
$n \times m$	135 × 8	155 × 8	175 × 8	195 × 8	215 × 8	235 × 8	255 × 8	275 × 8
GA-based method	5453.5 (5453.5)	7074.1 (7074.1)	8723.0 (8723.0)	11309.3 (11309.3)	14440.2 (14440.2)	18033.6 (18033.6)	24316.7 (24316.7)	36745.1 (36745.1)
MCTS-based method	3071.4 (1829.2)	4009.1 (2312.7)	4915.4 (2819.0)	6480.0 (3629.8)	8491.3 (4538.6)	10446.7 (5749.2)	13502.4 (7503.6)	17463.9 (10129.3)

The values $x(y)$ in this table denote that the mean TCPU time and the mean TR time are x and y, respectively.

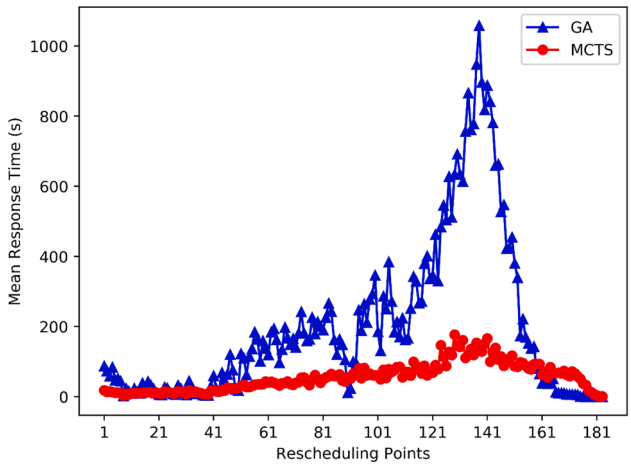

Fig. 5. The mean response time comparison of MCTS-based method and GA-based method at each rescheduling point in the instance D18.

It can be found that compared to the completely reactive scheduling methods, the GA-based rescheduling method and the proposed MCTS-based rescheduling method can finish all operations in a much shorter time. For each instance, although the result of the MCTS-based rescheduling method is slightly worse than the GA-based rescheduling method, the TCPU time and the total response time to all dynamic events of the former is much less than that of the latter, which is more suitable to be applied in practical production, especially for the large-scale scheduling problem with frequent dynamic events.

5.2.2. Analysis of the response time

It is worth pointing out that the response time to dynamic events plays a crucial role in solving dynamic scheduling problems in the real manufacturing. In this paper, the computation time to generate a new rescheduling scheme at each rescheduling point is taken as the response time. The mean response time comparison of MCTS-based method and GA-based method at each rescheduling point in the instance D18 is plotted in Fig. 5, which shows that the response time of MCTS-based method is much shorter than that of GA-based method at most of the rescheduling points. The difference is more obvious with the increase of the number of unprocessed operations. The slightly longer response time of MCTS-based method at the last few rescheduling points may be due to the greater number of remaining unprocessed operations. In addition, the proposed MCTS-based method has a more stable response time throughout all the rescheduling points, even in the case of a very large number of unprocessed operations.

6. Conclusions and future work

Most of the literature on production scheduling focuses on static scheduling problems, while dynamic factors are seldom considered due to their complexity and difficulty. This paper studied a dynamic flexible job shop scheduling problem which takes four dynamic events into account, namely, new jobs arrival, machine breakdown, job cancellation and change in the processing time of operations.

In dynamic scheduling problem, it is very important to respond to random dynamic events and generate an acceptable rescheduling scheme in a short time. However, some traditional intelligent optimization algorithms often take a long time to generate a satisfactory schedule, especially for large-scale FJSPs, which makes them not directly applicable to actual production scheduling scenarios. In order to contribute to the literature, this paper introduced a new rescheduling method based on MCTS, which can quickly generate an effective schedule for the proposed DFJSP. In this way, by comparing with these traditional rescheduling methods, it can be realized that the proposed method provides an acceptable solution to the DFJSP in terms of solution quality and computation time.

The MCTS algorithm is firstly used to solve the DFJSP. Unlike the traditional rescheduling methods which generate complete rescheduling scheme at each rescheduling point, the MCTS-based rescheduling method is applied to generate a partial rescheduling scheme corresponding to the subsequent specified time window in a short time. This method can greatly reduce the real-time response time to dynamic events, so it can be better applied in the actual scheduling environments. Since the proposed method considers dynamic factors and is able to provide efficient and effective solution even in large scale problems with frequent dynamic events, it can be applied to practical manufacturing systems.

We believe that the MCTS-based method can be considered as a new useful approach to obtain efficient and effective solution for different scheduling problems. In future research, it is worthwhile to study many other performance objectives or multi-objective for DFJSPs. There are some other optimization techniques to improve the performance of MCTS which deserve further study. Moreover, the proposed method can be considered to deal with other different production scheduling environments such as flow shop and parallel machines shop.

CRediT authorship contribution statement

Kexin Li: Conceptualization, Methodology, Software, Formal analysis, Writing - original draft. **Qianwang Deng:** Supervision, Funding acquisition, Resources, Writing - review & editing. **Like Zhang:** Investigation, Validation, Writing - review & editing. **Qing Fan:** Project administration, Funding acquisition. **Guiliang Gong:** Project administration, Funding acquisition. **Sun Ding:** Software.

Acknowledgments

The authors would like to express heartfelt thanks to the editor and anonymous reviewers for their valuable comments and suggestions.

This work was supported by the National Key R&D Program of China [grant numbers 2020YFB1712100, 2018YFB1701400]; the National Natural Science Foundation of China [grant numbers 61973108, 72001217]; the State Key Laboratory of Advanced Design and Manufacturing for Vehicle Body, Hunan University [grant number 71775004]; the Nature Science Foundation of Changsha [grant number kq2007033]; and the State Key Laboratory of Construction Machinery [grant number SKLCM2019-03].

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A

A summary of the parameters used to design the DFJSP instances in our experiments is presented in Table A.1. The number of available machines is 8, and each machine is assigned a different mean time between failure (MTBF) and mean time to repair (MTTR). The MTBF and MTTR of eight machines are given in Table A.2.

Table A1
Summary of the parameters used in the design of DFJSP instances.

Characteristics	Specifications
Size	$m = 8$ machines
Change in the processing time of operations	$U[1, 10]$
Machine breakdowns	MTBF = $U[50, 70]$ MTTR = $U[10, 20]$
Distribution of TBF	Exponential distribution with the MTBF
Distribution of TTR	Exponential distribution with the MTTR
The number of operations for each new job	$U[1, m + 2]$
The number of optional machines for each new operation	at random from the set {1, 2, …, m}
The processing time of each new operation	$U[1, 10]$
The mean time between job arrival (MTBJA)	20
Distribution of the time between job arrival (TBJA)	Exponential distribution with the MTBJA
The mean time between job cancellation (MTBJC)	60
Distribution of the time between job cancellation (TBJC)	Exponential distribution with the MTBJC
The number of new jobs per arrival	5
The number of jobs per cancellation	1

$U[a, b]$ denotes a number generated uniformly at random from the interval of $[a, b]$.

Table A2
The MTBF and MTTR of eight machines.

Machine number	1	2	3	4	5	6	7	8
MTBF	58	68	62	58	69	59	66	51
MTTR	19	10	14	13	17	11	20	14

References

Adibi, M. A., Zandieh, M., & Amiri, M. (2010). Multi-objective scheduling of dynamic job shop using variable neighborhood search. *Expert Systems with Applications, 37*(1), 282–287. https://doi.org/10.1016/j.eswa.2009.05.001

Asta, S., Karapetyan, D., Kheiri, A., Ozcan, E., & Parkes, A. J. (2016). Combining Monte-Carlo and hyper-heuristic methods for the multi-mode resource-constrained multi-project scheduling problem. *Information Sciences, 373*, 476–498. https://doi.org/10.1016/j.ins.2016.09.010

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research, 41*(1–4), 157–183. https://doi.org/10.1007/bf02023073

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., … Colton, S. (2012). A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and Ai in Games, 4*(1), 1–43. https://doi.org/10.1109/tciaig.2012.2186810

Cao, Z., Zhou, L., Hu, B., & Lin, C. (2019). An adaptive scheduling algorithm for dynamic jobs for dealing with the flexible job shop scheduling problem. *Business & Information Systems Engineering, 61*(3), 299–309. https://doi.org/10.1007/s12599-019-00590-7

Chou, J.-J., Liang, C.-C., Wu, H.-C., Wu, I. C., & Wu, T.-Y. (2015). A new MCTS-based algorithm for multi-objective flexible job shop scheduling problem. In *2015 Conference on Technologies and Applications of Artificial Intelligence* (pp. 136–141).

Chryssolouris, G., & Subramaniam, V. (2001). Dynamic scheduling of manufacturing job shops using genetic algorithms. *Journal of Intelligent Manufacturing, 12*(3), 281–293. https://doi.org/10.1023/a:1011253011638

Church, L. K., & Uzsoy, R. (1992). Analysis of periodic and event-driven rescheduling policies in dynamic shops. *International Journal of Computer Integrated Manufacturing, 5*(3), 153–163. https://doi.org/10.1080/09511929208944524

Furuoka, R., & Matsumoto, S. (2017). Worker's knowledge evaluation with single-player Monte Carlo tree search for a practical reentrant scheduling problem. *Artificial Life and Robotics, 22*(1), 130–138. https://doi.org/10.1007/s10015-016-0325-2

Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research, 1*(2), 117–129. https://doi.org/10.1287/moor.1.2.117

Gelly, S., & Silver, D. (2007). Combining online and offline knowledge in UCT. In *Proceedings of the 24th International Conference on Machine Learning* (pp. 273–280).

Gelly, S., & Silver, D. (2011). Monte-Carlo tree search and rapid action value estimation in computer Go. *Artificial Intelligence, 175*(11), 1856–1875. https://doi.org/10.1016/j.artint.2011.03.007

Gholami, M., & Zandieh, M. (2009). Integrating simulation and genetic algorithm to schedule a dynamic flexible job shop. *Journal of Intelligent Manufacturing, 20*(4), 481–498. https://doi.org/10.1007/s10845-008-0150-0

Kocsis, L. & Szepesvari, C. (2006). Bandit based Monte-Carlo planning. In J. Furnkranz, T. Scheffer & M. Spiliopoulou (Eds.), Proceedings machine learning: Ecml 2006 (Vol. 4212, pp. 282–293).

Kundakci, N., & Kulak, O. (2016). Hybrid genetic algorithms for minimizing makespan in dynamic job shop scheduling problem. *Computers & Industrial Engineering, 96*, 31–51. https://doi.org/10.1016/j.cie.2016.03.011

Li, X., & Gao, L. (2016). An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem. *International Journal of Production Economics, 174*, 93–110. https://doi.org/10.1016/j.ijpe.2016.01.016

Liu, B., Fan, Y., & Liu, Y. (2015). A fast estimation of distribution algorithm for dynamic fuzzy flexible job-shop scheduling problem. *Computers & Industrial Engineering, 87*, 193–201. https://doi.org/10.1016/j.cie.2015.04.029

Mehta, S. V., & Uzsoy, R. (1999). Predictable scheduling of a single machine subject to breakdowns. *International Journal of Computer Integrated Manufacturing, 12*(1), 15–38. https://doi.org/10.1080/095119299130443

Nie, L., Gao, L., Li, P., & Li, X. (2013). A GEP-based reactive scheduling policies constructing approach for dynamic flexible job shop scheduling problem with job release dates. *Journal of Intelligent Manufacturing, 24*(4), 763–774. https://doi.org/10.1007/s10845-012-0626-9

Nie, L., Gao, L., Li, P., & Shao, X. (2013). Reactive scheduling in a job shop where jobs arrive over time. *Computers & Industrial Engineering, 66*(2), 389–405. https://doi.org/10.1016/j.cie.2013.05.023

Ouelhadj, D., & Petrovic, S. (2009). A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling, 12*(4), 417–431. https://doi.org/10.1007/s10951-008-0090-8

Ozturk, G., Bahadir, O., & Teymourifar, A. (2019). Extracting priority rules for dynamic multi-objective flexible job shop scheduling problems using gene expression programming. *International Journal of Production Research, 57*(10), 3121–3137. https://doi.org/10.1080/00207543.2018.1543964

Rajabinasab, A., & Mansour, S. (2011). Dynamic flexible job shop scheduling with alternative process plans: An agent-based approach. *International Journal of Advanced Manufacturing Technology, 54*(9–12), 1091–1107. https://doi.org/10.1007/s00170-010-2986-7

Rangsaritratsamee, R., Ferrell, W. G., & Kurz, M. B. (2004). Dynamic rescheduling that simultaneously considers efficiency and stability. *Computers & Industrial Engineering, 46*(1), 1–15. https://doi.org/10.1016/j.cie.2003.09.007

Reddy, M. B. S. S., Ratnam, C., Rajyalakshmi, G., & Manupati, V. K. (2018). An effective hybrid multi objective evolutionary algorithm for solving real time event in flexible job shop scheduling problem. *Measurement, 114*, 78–90. https://doi.org/10.1016/j.measurement.2017.09.022

Sha, D. Y., & Liu, C. H. (2005). Using data mining for due date assignment in a dynamic job shop environment. *International Journal of Advanced Manufacturing Technology, 25*(11–12), 1164–1174. https://doi.org/10.1007/s00170-003-1937-y

Shen, X.-N., & Yao, X. (2015). Mathematical modeling and multi-objective evolutionary algorithms applied to dynamic flexible job shop scheduling problems. *Information Sciences, 298*, 198–224. https://doi.org/10.1016/j.ins.2014.11.036

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., … Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature, 529*(7587), 484-+. https://doi.org/10.1038/nature16961

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., & Hassabis, D. (2017). Mastering the game of Go without human knowledge. *Nature, 550*(7676), 354. https://doi.org/10.1038/nature24270

Su, N., Zhang, M., Johnston, M., & Tan, K. C. (2013). A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem. *IEEE Transactions on Evolutionary Computation, 17*(5), 621–639.

Teymourifar, A., Ozturk, G., Ozturk, Z. K., & Bahadir, O. (2020). Extracting new dispatching rules for multi-objective dynamic flexible job shop scheduling with limited buffer spaces. *Cognitive Computation, 12*(1), 195–205. https://doi.org/10.1007/s12559-018-9595-4

Vieira, G. E., Herrmann, J. W., & Lin, E. (2003). Rescheduling manufacturing systems: A framework of strategies, policies, and methods. *Journal of Scheduling, 6*(1), 39–62. https://doi.org/10.1023/a:1022235519958

Vinod, V., & Sridharan, R. (2008). Scheduling a dynamic job shop production system with sequence-dependent setups: An experimental study. *Robotics and Computer-Integrated Manufacturing, 24*(3), 435–449. https://doi.org/10.1016/j.rcim.2007.05.001

Waledzik, K., & Mandziuk, J. (2018). Applying hybrid Monte Carlo tree search methods to risk-aware project scheduling problem. *Information Sciences, 460*, 450–468. https://doi.org/10.1016/j.ins.2017.08.049

Wang, L., Luo, C., & Cai, J. (2017). A variable interval rescheduling strategy for dynamic flexible job shop scheduling problem by improved genetic algorithm. *Journal of Advanced Transportation, 2017*(3), 1–12. https://doi.org/10.1155/2017/1527858

Wu, T.-Y., Wu, I. C., & Liang, C.-C. (2013). Multi-objective flexible job shop scheduling problem based on Monte-Carlo tree search. In *2013 conference on technologies and applications of artificial intelligence* (pp. 73–78).

Xu, B., Mei, Y., Wang, Y., Ji, Z., & Zhang, M. (2020). Genetic programming with delayed routing for multiobjective dynamic flexible job shop scheduling. *Evolutionary Computation, 1–31*. https://doi.org/10.1162/evco_a_00273

Zadeh, M. S., Katebi, Y., & Doniavi, A. (2019). A heuristic model for dynamic flexible job shop scheduling problem considering variable processing times. *International Journal of Production Research, 57*(10), 3020–3035. https://doi.org/10.1080/00207543.2018.1524165

Zandieh, M., & Adibi, M. A. (2010). Dynamic job shop scheduling using variable neighbourhood search. *International Journal of Production Research, 48*(8), 2449–2458. https://doi.org/10.1080/00207540802662896

Zhang, F., Mei, Y., & Zhang, M. (2019). Evolving dispatching rules for multi-objective dynamic flexible job shop scheduling via genetic programming hyper-heuristics. In *2019 IEEE congress on evolutionary computation* (pp. 1366–1373).

Zhang, Y., Wang, J., & Liu, Y. (2017). Game theory based real-time multi-objective flexible job shop scheduling considering environmental impact. *Journal of Cleaner Production, 167*, 665–679. https://doi.org/10.1016/j.jclepro.2017.08.068

Zhou, H., Pang, J., Chen, P.-K., & Chou, F.-D. (2018). A modified particle swarm optimization algorithm for a batch-processing machine scheduling problem with arbitrary release times and non-identical job sizes. *Computers & Industrial Engineering, 123*, 67–81. https://doi.org/10.1016/j.cie.2018.06.018