A MCTS-based approach for dynamic job shop scheduling problem

Abstract

In most real manufacturing environments, schedules are usually nor infeasible or optimal with the presence of various unexpected disruptions, so it is of significant importance to respond to various disruptions quickly(且现有文献缺少解决该类扰动的高效方法). In this paper, we employ Monte Carlo Tree Search (MCTS) in combination with commonly used dispatching rules to devise strategies for the dynamic job shop scheduling problem with a single or a combination of different types of real-time events, including new job arrival, machine breakdowns, order cancellation and change in the processing time of an operation. To deal with the dynamic problems, an event-driven policy is selected. To enhance the efficiency and effectiveness of the rescheduling method, several optimization techniques are used to improve MCTS. The novelty of the proposed method is that a high-quality reschedule can be generated timely and quickly when various real-time events occur. The performance measures investigated respectively are: makespan, and machine utilization. Because it is difficult to be expressed by the mathematical model, a simulator is proposed to generate dynamic job shop problems including number of orders, number of machines and various dynamic events. Numerical experiments have been designed to test and evaluate the performance of the proposed method. This proposed method has compared with some classical dispatching rules which have widely been used in the literature. The experimental results illustrate that the proposed method is more effective and practical in dynamic job shop problems.

1. Introduction

In current manufacturing systems, production is a dynamic process with many unexpected events and continuously emergs new requirements. The requirement of rescheduling in response to unexpected changes is commonplace in modern flexible decision making and manufacturing systems. However, in dynamic environments, managers and production planners must not only generate high quality schedules, but also react quickly to unexpected events. (调度问题成为众多学者研究的热门话题，但大部分都是假设所有条件已知，缺少动态扰动事件的考虑。然而，实际的生产系统经常伴随不确定性事件的发生，导致原有的调度计划失效或非最优。因此，求解动态调度问题的研究开始引起学者们的注意)

The first study in dynamic job shop problem(DJSP) was published by Nelson, Holloway, and Mei-Lun Wong (1977). Nouiri, Bekrar et al. (2018) who proposed a scheduling system for a job shop with intermittent job arrivals. They implemented a multi-pass procedure by generating schedules periodically. Karsiti, Cruz,and Mulligan (1992) proposed heuristic rules which were used to determine the job-to-machine routing and the job sequencing. Fang and Xi (1997) used a hybrid of genetic algorithms and dispatching rules to solve JSSP with sequence-dependent set-up time and due date constraints under job arrivals, machines breakdown, due date change events. Holthaus (1999) presented a simulation-based analysis of dispatching rules for DJSSP under machine breakdowns with mean and maximum flow time, variance of flowtime, mean and maximum tardiness, percentage of tardy jobs and variance of tardiness. Kutanoglu (1999) tested new scheduling techniques using simulation under new order arrivals. Aydin and Öztemel (2000) presented an intelligent agent-based dynamic scheduling system under new job arrivals. Sabuncuoglu and Bayiz (2000) tested different scheduling policies under machine breakdowns in dynamic job shop system and measured the effect of system size, type of work allocation on the system performance, mean tardiness and makespan criteria. Subramaniam et al. (2000) suggested three machine selection rules for DJSSP. Piramuthu, Shaw, and Fulkerson (2000) presented an adaptive scheduling policy for dynamic manufacturing system scheduling using information obtained from snapshots of the system at various points in time. Qi, Burns, and Harrison (2000) proposed parallel multi-population genetic algorithm for DJSSP. Kutanoglu and Sabuncuoglu (2001) proposed iterative simulation-based scheduling mechanisms for manufacturing systems that are operated in dynamic and stochastic environments. Dewan and Joshi (2002) developed an auction mechanism that can be used to make a schedule within a distributed decision-making environment. Rangsaritratsamee, Ferrell, and Kurz (2004) suggested a genetic local search procedure to construct schedules at each periodic rescheduling point for improving efficiency and schedule instability measures with new job arrivals. Dominic, Kaliyamoorthy, and Kumar (2004) proposed efficient dispatching rules for DJSSP by combining different dispatching rules using simulation. Liu, Ong, and Ng (2005) applied a metaheuristic to solve DJSSP. Kang, Feng Zhang, and Qing Yang (2007) considered dynamic events such as new job arrival, machine , breakdown ,and order cancellation in the job shop scheduling problem with the new structure of multi-agent system which is combined with ant colony optimization algorithm. Suwa and Sandoh (2007) proposed a new when-to-schedule policy in reactive scheduling under job shop environment with machine breakdowns. Xiang and Lee (2008) presented a multi-agent system equipped with ant colony intelligence to provide an efficient scheduling with sequence-dependent setups, and various dynamic disturbances. Vinod and Sridharan (2008) developed a discrete event simulation model of a job shop system in which set-up times are sequence dependent and new order arrival is considered as dynamic events. Zhou, Nee, and Lee (2009) used ant colony optimisation in dynamic job shop with event driven scheduling strategy and new job arrivals and compared the results with the results of dispatching rules. Vinod and Sridharan (2009) developed regression-based models for scheduling in job shop environment with sequence-dependent set-up times. Zandieh and Adibi (2010) presented a variable neighborhood search method for a multi-objective DJSSP with random job arrivals and machine breakdowns under event-driven scheduling strategy. Their multi-objective performance measure consisted of two efficiency criteria (makespan and tardiness). The proposed method was compared with some common dispatching rules using a simulated job shop under varied conditions. Vinod and Sridharan (2011) analysed the effects of due-date assignment methods and scheduling rules on the performance of the dynamic job shop production system using a discrete-event simulation model. Zhang, Gao, and Li (2013) used hybrid scheduling strategy with a hybrid genetic algorithm and tabu search for multi-objective DJSSP under random job arrivals and machine breakdowns to achieve two performance measures, which contain the schedule efficiency and the schedule instability and compared their results with the results from dispatching rules. Scholz-Reiter, Hildebrandt, and Tan (2013) presented a new scheduling method that combined shifting bottleneck procedure and the variable neighbourhood search for JSSP under new job arrivals to minimize job tardiness and constructed the schedule with rolling time horizon procedure. They compared the results with dispatching rules. Lu and Romanowski (2013) described multicontextual dispatching rules for JSSP with dynamic order arrival. Qiu and Lau (2013) presented an idiotypic network model whose antibody’s action part is the dispatching rule for solving the multi-objective DJSSP and they demonstrated the capability and efficiency of the proposed model. Sarker et al.

Monte Carlo Tree Search (MCTS) is a best-first search method to approach real results by Monte-Carlo sampling (Coulom 2007). It has been successfully and popularly used in the area of game playing, especially computer Go (Silver, Huang et al. 2016, Silver, Schrittwieser et al. 2017).

It can be applied to any Markov Decision Process (MDP), for which generative model exists [21]. So it can be useful in the wide research area of decision-making and decision-support [14], especially in stochastic problems with very large or infinite state spaces, for which many traditional algorithms and reinforcement learning approaches prove inapplicable.

So far, there have been few attempts at applying MCTS to scheduling problems (Pellier, Bouzy et al. 2010, Asta, Karapetyan et al. 2016, Furuoka and Matsumoto 2017), and its employment in project scheduling is a new idea, not yet significantly explored. Two published paper (Lu, Chiu et al. 2016) (Wu, Wu et al. 2013) successfully uses MCTS to solving the Flexible Job-shop scheduling problem (FJSP), a complex combinatorial optimization problem and well known as an NP-hard. They deal with fully deterministic problems, while we concentrate mainly on the dynamic production scheduling.

Despite the significant progress achieved by researchers, the following research questions are still open in applying job shop scheduling methods to the real-time manufacturing shop floor with the increasing complexity of production processes, the unpredictability of the production exceptions, etc. These questions are summarized as follows.

1. Almost all studies assume that only one type of uncertainty occurs at a time In the literature, there are few research papers which handle various dynamic events such as new order arrivals, machine breakdowns, order cancellations, change in the processing time of an operation simultaneously under set-up times, machine capacity. However, in an actual dynamic production system, many different types of uncertain events may occur simultaneously or in a short period of time, and the effects of various uncertain events on the production system vary. Once this occurs, the combined effects of various uncertainties on the production system will cause greater damage and interference to the production system. In view of this situation, it is necessary to propose an efficient algorithm that can respond to various real-time events quickly.
2. In terms of algorithms in solving DJSP, heuristic priority rules are simple, fast and easy to implement, but cannot guarantee the quality of solutions. Intelligent algorithms such as widely used genetic algorithms usually require a series of improvement and optimization operations based on the characteristics of the problem, and the generalization ability of these optimization operations whose design is complicated and cumbersome is limited, . Usually it will take a long time to generate a high-quality solution, which is not conducive to rapid response when uncertain events occur. So it is necessary to find a more general search algorithm without designing special optimization operators, which can respond to various uncertain events timely for DJSP with different scales.
3. In practice, large deviations or changes to the sequence will occur when the real-time events disrupt the initial schedule, and the schedule after the rescheduling point often become infeasible or inefficient, so there is no need to search for the schedules of all operations, So (与之相反)we can search for the schedule while processing according to the already generated schedule in a dynamic manufacturing system.

To address the above questions, in this study, a MCTS-based event-driven rescheduling method (MCTS-ERM) was presented to address the above newly emerged problems. As a complex dynamic problem, (DJSP) can be represented as a Markov Decision Process with explicitly available generative model, it is a natural candidate for MCTS. In our method, when real-time events occur, instead of generating a complete schedule, we determine the processed order for all remaining operations in turn while the job shop start to process according to the schedule that has been generated. The complete rescheduling strategy, which regenerates a new schedule from scratch, is used at each rescheduling point. Experimental results show that our method can react quickly to unexpected events and achieve high solutions in a simulated dynamic shop. To our best knowledge, in the literature reported, MCTS has not yet been adopted to regenerate new schedules in a reactive way when shop environments change.

This paper is organized as follows. The literature review is presented in Section 2. The proposed MCTS-based algorithm is presented in Section 3. The problem is defined in Section 4. In Section 5, the experimental design and results are discussed. Finally, Section 6 gives the conclusions.

1. Monte Carlo Tree Search

Monte Carlo Tree Search is an iterative method. With the algorithm is running, a search tree of the problem is built in memory, and successively becomes better at accurately estimating the values of the candidate moves. The more time the program runs, the stronger the program execute. It repeats the following four phases ~~as presented in figure1~~.

1. Selection: Among various selection policies, the predominant one – UCT (Upper Confidence bounds applied to Trees) (Kocsis and Szepesvari 2006), is employed in our selection phase. Starting from root node, recursively choose the best child node based on the following formula until a leaf is reached.



where *Q*(*s, a*) is the average winrate (or value) of performing action *a* in state *s* so far, *A*(*s*) denotes the set of all actions legal in state *s*, *N*(*s*) denotes current number of visits to state *s*, and *N*(s*, a*) is current number of times action a has been visited in this state. The tree search tends to exploit (choosing action with the highest expected reward) when *C* is small, while it tends to explore (trying action with the fewest visits) when *C* is large.

1. Expansion: When a leaf node is reached, expand a new node from the leaf node and add it to the tree.
2. Playout: Perform random rollouts from the newly expanded node until there are no more actions except for the final payoff.
3. Backpropagation: the evaluation values is propagated up the path selected in the tree, updating the *Q*(*s, a*), *N*(s*, a*) and *N*(*s*) values in each node accordingly.

An iteration of these four phases is called a simulation.

1. Simulator for DJSP

In order to validate the effectiveness and efficiency of our proposed method, a realistic job shop has been simulated. In the practical manufacture system, there are many real-time events, such as new job arrival, machine breakdown. In this study, we consider four common types of real-time events: new order arrival, machine breakdown, order cancellation and change in the processing time of an operation.

Since these real-time events occur randomly, a simulator is required to simulate different situations. The simulator, which contains new job arrival, machine breakdowns and repairs, order cancellation and change in the processing time of an operations, is used to generate the disturbances.

3.1 Technological assumptions

Dynamic job shop scheduling problem subjects to the following assumptions (Zhang, Gao et al. 2013):

1. Each machine can perform only one operation of any job at a time.
2. An operation of a job can be performed by only one machine at a time.
3. All machines are available at time 0.
4. Once an operation has been processed on a machine, it must not be interrupted except due to machine breakdown. If an operation is interrupted by a machine breakdown, the remained processing time is equal to total processing time minus the completed processing time.
5. An operation of a job cannot be performed until its preceding operations were completed.
6. There is no flexible routing for each job.
7. Processing time of all operations and the number of operable machines are known in advance. But there will be a change in the operation processing time and machines can breakdown.

3.2 Details for various real-time events

In the following section, the details and effects of various disruptions are discussed in detail:

1. New jobs arrive at the system dynamically over time: In dynamic job shops, the distribution of job arrivals process closely follows a Poisson distribution. Hence, the time between job arrivals closely follows an Exponential distribution (Rangsaritratsamee, Ferrell et al. 2004) (Sha and Liu 2005) (Vinod and Sridharan 2008). When rescheduling is triggered, there are four types of job sets: finished job set, being processed job set, unprocessed job set and new job set. When the new jobs arrive, the jobs are pushed into the new job set.
2. Machine breakdown and repair: the time between two machine failures and the repair time are assumed to follow an Exponential distribution (Zandieh and Adibi 2010). The mean time between failure (MTBF) and the mean time to repair (MTTR) are two parameters related to machine breakdown. When a machine breaks down, the ongoing operation is assumed to be not disrupted and machine repairing is carried out after the ongoing operation is completed. The beginning and completion time of non-started operations are arranged depending on the repair time of the breakdown machine.
3. Order cancellation: Some customers may cancel their orders. After order cancellation, the remaining operations of this order are deleted from the schedulable task list and the rescheduling process is performed. In dynamic job shops, the distribution of job arrivals process is assumed to closely follow a Poisson distribution too. Hence, the interval between a pair of adjacent job arrivals closely follows an Exponential distribution.
4. Change in the processing time of an operation: Before the workshop starts processing, the processing time of some operations will be changed due to the change of order tasks. Then schedule is constructed with the new processing times.

In our method, the complete schedule is not generated before the start of processing, the schedule stability is not required to be optimized. Makespan is used as the optimization objective, which is also an optimization objective widely used in most literatures. It is denoted by C, and it can be calculated according to



where *Ci* is the completion time of each job.

1. The proposed method for DJSP

In this research, a MCTS-based rescheduling method has been presented for continuous processing in a dynamic job shop floor.

4.1 The flowchart of the proposed method

In the most of practical manufacturing, a successful implementation of a scheduling system usually requires updating the problem condition, redistributing the processing sequence of the operations on the machine at each rescheduling point. The general process of the proposed approach is summarized in Figure 1.

At the beginning of the scheduling system, all machines are available at time 0. After update the system state, six complete schedules are generated by using six common dispatching rules (SPT, LPT, SRPT, LRPT, FIFO, LIFO), and the schedule with the smallest makespan is selected to determine the next operation to be processed by each machine in the current state. Then, the shop floor begins execution of the operations, meanwhile the modified MCTS is performed to determine the processing sequence of the remaining operations in turn until the next rescheduling point. When one or several real-time events occur at the next rescheduling point, the reschedule will be triggered. The system state, which contains the every machine’s available time, unprocessed job set, being processed job set, processed job set, new job set and the completion time of the previous operation of each job, is updated. Then the above procedure is repeated until all operations have been processed.

At each rescheduling point, the machine can be classified into three categories: the machine busy (an operation is being processed on this machine) , the machine idle and the machine broken. In the case of the machine busy, the machine available time will be assumed the completed time of the operation being processed on the machine. In the case of the machine idle, the machine available time will be the rescheduling point. In the case of the machine broken, the machine available time will be the time the machine is repaired.

Based on the above design and description, our method can generate a reschedule quickly and timely when real-time events occur, and it can be summarized as the following steps:

1. Update the system state, and six dispatching rules mentioned above are used to generate six complete schedules, then the one with the smallest makespan is selected to determine the operation to be processed next on each machine.
2. The job shop floor begins to execute operations according to schedule generated by the dispatching rule, meanwhile the system state is updated again, then the modified MCTS is performed to generate the schedules for the remaining operations until the next rescheduling point, and each machine continues processing according to the schedule generated by dispatching rule or the modified MCTS.
3. When dynamic events occur, if the modified MCTS is still running, it will be stopped. Then repeat Step 1 and Step 2.
4. Repeat Step 1, Step 2 and Step 3 until all operations have been processed.



Figure1. Flowchart of the proposed MCTS-based method for DJSP.

4.2 The proposed MCTS-ERM for DJSP

The MCTS-based rescheduling method will be performed to generate a high-quality reschedule for continuous processing in a dynamic environment at each rescheduling point. In order to implement the proposed method, some detailed designs and descriptions have been made, which contain problem transformation, modified MCTS, mechanism of cooperation between MCTS and dispatching rules, decoding of the operation sequence, node selection strategy after N MCTS simulations, update of the job shop state.

4.2.1 Problem Transformation

The follows describes how to map JSP to MCTS process, that is, run MCTS to determine the processing order of each operation on the assigned machine in job shop problem.

Assume that a simple data structure with root node *r*. Every node in the tree is associated with a candidate hybrid label pair (operation, machine). Every node *n* other than the root node *r* has a unique parent node. Every non-leaf node has d ≥ 1 child nodes. For simplicity of discussion, we use the following notations. *Nop*denotes the total number of operations, [*j, i*]: the operation *Oj,i*, [*j, i, k*]: O*j,i* is processed on machine k, [*j, i, k, seq*]: O*j,i* is the *seq*’th operation processed on machine *k*. Note that [*j, i, k, seq*] is also called a move in this paper.

The follows present the mapping procedure based on a modified MCTS, which is described in Section 4.2.2 for detail, and tree techniques for improving the efficiency of MCTS are also included in Section 4.2.2. For detailed description, we use a 4×2 job shop problem instance listed in Table 1 to illustrate the mapping procedure step by step. The construction of search tree topology is shown in Figure 2. The mark on each child node in the figure is the operation waiting for processing.

Table 1

An instance of the JSP with 4 jobs, 2 machines.

|  |  |  |
| --- | --- | --- |
| Jobs | Machine sequences and processing times | |
| J1 | M3(2) | M1(8) |
| J2 | M2(6) | M3(5) |
| J3 | M1(7) | M5(8) |
| J4 | M4(4) | M3(5) |



Figure 2. Transformation to tree search structure for the sample job shop problem.

1. Initialize schedule and the root node of MCTS
   1. Let *S* be an empty schedule and no operations are scheduled yet.
   2. Set r as the initial root of MCTS
2. Repeat the following steps *Nop* times to get a complete schedule.

## Perform MCTS with *Ns* simulations.

## After the MCTS in the previous substep is completed, choose the child node using the strategies proposed in Section 4.2.5 as the best move, called [*j,i,k,seq*]. For example, after *Ns* MCTS simulations, we finally chose the node marked by *O*(*2, 1*). Since *O*(*2, 1*) is the first operation of job 2 and the first operation to be processed on the machine 3 (The assigned machine of *O*(*2, 1*)), move of the chosen child node is [*2, 1, 3, 1*].

## Decode [*j, i, k, seq*] into S with the decoding procedure proposed in Section 4.2.4.

## Set the chosen node as the new root node of MCTS for next iteration.

In Step 2 of this algorithm, when *Ns* MCTS simulations are completed, it finds the best move or moves and decode them into *S.* After *Nop* iterations, all operations will be scheduled, then we will get a completed schedule.

4.2.2 Our modified MCTS

Even for the job shop problem, its search space is too huge to be exhausted. In order to improve the efficiency of MCTS and choose a good move after *Ns* simulations, we intend to adopt the following tree optimization techniques:

* 1. Subtree Keeping Policy

Every time after running *Ns* MCTS simulations, we choose the best child node of the root node as the new root node, then instead of building a subtree based on the chosen node from scratch, the subtree of the chosen child node will be retained for next iteration. So information obtained by previous simulations, including the number of visited times and scores of each node in the retained subtree, will be reused. This plolicy can deepen the search depth and widen the search width of MCTS and get a more accurate score for the child node of the new root node.

* 1. Rapid Action Value Estimates Heuristic (RAVE)

Based on the underlying idea of the all-moves-as-first (AMAF) heuristic (Bouzy and Helmstetter 2004), Gelly and Silver (2011) proposed a kind of AMAF heuristic, called Rapid Action Value Estimates Heuristic (RAVE). RAVE provides a simple way to share knowledge between related nodes in the search tree, resulting in a rapid, but biased estimate of the action values. This biased estimate can often determine the best move after just a handful of simulations, and can be used to significantly improve the performance of the search algorithm. Thus, we consider additionally the RAVE estimation when doing selection, and the weighted sum *Q*͙(*s, a*) of the MC value *Q*(*s, a*) and the AMAF value *Qrave* (*s, a*) is as follows.



where *β* is the bias between the Monte-Carlo and RAVE value, the optimal *β* is calculated as follows:



where *b* is a constant, *N*(*s, a*)and *Nrave*(*s, a*)is Monte-Carlo visit count and the RAVE visit count of the current node’s move a respectively.

Similarly to the UCT algorithm, an exploration bonus is also incorporated to keep the balance between exploration and exploitation, the final formula is as follows:



* 1. Prior Knowledge

In order to make full use of the results of previous simulations, these results will be recorded in a global array *Kprior* (Wu, Wu et al. 2013).*Kprior* is a two-dimensional array whose size is *Nop*×*Njob,* indexed by ([*j, i*], *k*, *seq*). The second dimension is *Njob* because there are *Njob* processed orders for each operation on the assigned machine. It records the average values and visit counts of moves obtained so far. When expanding nodes, we first give fixed initial values 0.5 to the newly expanded nodes, then consider additionally the average values of these newly expanded nodes in *Kprior*.

*Kprior* is constructed at the beginning of the algorithm. The initial value is set to 0.5 and visit count is set to 1. Once we get a complete schedule G and its evaluation value *v* after a MCTS simualtion is completed, moves of this schedule is updated to *Kprior* as following steps:

* 1. Let m be a move in G.
  2. *Avg* and *visit* denote the average value and visit count of *Kprior* (*m*) respectively.
  3. Update *Avg*



* 1. Update *visit*



* 1. If G has no other operations, stop; otherwise, let d be the next move of G.

The newly expanded nodes can get initial value according to *Kprior*:



Based on the above optimization techniques, we made some modifications to the basic MCTS as follows:

1. Selection

Starting from the root node, node is recursively chosen to maximize *Q͙Z*(*s, a*) until a leaf node is reached. In this paper, we only consider makespan when doing selection. The reasons are: (1) In JSP, makespan is the most common criterion (Chiang and Lin 2012). (2) The smaller makespan implies higher machine utilization (Kashan, Karimi et al. 2010)

So in our MCTS, the final payoff is evaluated by the following evaluation function:



where best(*CM*)is the best makespan found so far, the evaluation value will become big if the makespan of a schedule is small.

1. Expand

When a leaf node is reached, we add the new node with the largest *InitValue* to the search tree.

1. Playout

Starting from the newly-added node, repeat to choose a move randomly until there are no more operations to choose.

1. Backpropagation

In this phase, updating the *Q*(*s, a*), *N*(*s, a*), *Qrave* (*s, a*) and *Nrave*(*s, a*) values in each node accordingly. In addition, we also update the global array *Aprior*.

Figure 4 is the Gantt chart for a 10×5 dynamic scheduling problem, which is used to describe the following designs in detail. The rescheduling point in Figure. 4 is *tr* = 106, and at *tr* = 106, a new job arrives, processing tasks of job 3 and job 5 are cancelled, and machine 2 is broken for 13 seconds.

4.2.3 Mechanism of cooperation between MCTS and dispatching rules

As shown in the Figure 3, after updating the system state at time A, six dispatching rules (SPT, LPT, MWKR, LWKR, FIFO, LIFO) are implemented to generate six complete reschedule immediately. Then, the schedule with the smallest makespan are chosen to determine the next operation to be processed for each machine, here *O*(6,3), *O*(10,1), *O*(8,2), *O*(4,4) and *O*(1,2) are to be processed next on six machines respectively. And we perform the modified MCTS to schedule all the remaining operations. Finally we got a reschedule whose makespan is 542s. We designed the cooperation mechanism based on the following reasons:

1. The schedule efficiency implies the high machine utilization. When a dynamic event occurs, the idle machine, such as machine 2can process the next operation immediately or after a short time, the busy machine, such as machine 3, 4, and 5 can continue to process the next operation without waiting for the reschedule to be generated. Therefore continuous processing is possible and it is beneficial to the improve machine utilization
2. The search time is reserved for MCTS simulation, it means that MCTS is performed while the job shop floor is executing operations according to the schedule generated by dispatching rule. And since the number of remaining operations is reduced, the search space and the search time are reduced.
3. The reason for selecting the schedule with the smallest makespan is to obtain a good "start" to generate a schedule with a small makespan. Because the processing sequence of the previous operations will affect the schedule of the subsequent operations, which further affects the makespan.



Figure 3. Gantt chart for a 10×5 dynamic scheduling problem.

4.2.4 Decoding

When an MCTS simulation is completed, an operation sequence is obtained by the Expand and Playout of MCTS. For the convenience of objective evaluation, the operation sequence should be decoded into the form of Gantt chart. Assume that we obtain an operation sequence after an MCTS simulation. And the operation sequence and the predefined processing time of each operation on the assigned machine are listed in Table 2. Figure 4 gives the Gantt chart of the operation sequence. Assume the rescheduling point in Fig 4 is *tr*= 0, and six operations *O*(6, 1), *O*(6, 2), *O*(4, 2), *O*(1, 1), *O*(4, 1), and *O*(1, 2) have been scheduled.

Table 2

Operation sequence and processing time of each operation on the assigned machine.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Operation sequence | *O*(10, 1) | *O*(9, 1) | *O*(1, 2) | *O*(10, 2) | *O*(9, 2) | *O*(11, 2) | *O*(6, 3) | *O*(4, 3) |
| Assigned machine | 3 | 3 | 1 | 4 | 4 | 3 | 2 | 2 |
| Processing time | 24 | 93 | 40 | 62 | 63 | 55 | 76 | 32 |

The process of constructing the Gantt chart in Figure. 4 is described as follows.

For the sake of simplicity, the start time of an operation is denoted by *ts*. First, take the first operation *O*(10, 1) in the operation sequence into account. Since the first operation *O*(10, 1) is the first operation of job 10 and its assigned machine 3 is idle, it is scheduled at *tr* = 0 for a processing time of 65 seconds. Then, since the second operation *O*(9, 1) is also assigned to machine 3. It can be processed until *O*(10, 1) (its previous operation in the same machine 3) has finished. Next, Since *O*(1, 2) is to be processed on machine 3 at the maximum value of the completion time of *O*(1, 1) and *O*(4, 1) (its previous operation in the same machine 6).The following operations in the operation sequence are scheduled to the assigned machines following the same method.



Figure 4. Decoded schedule of the chromosome in Table 2

It should be noted that the idle time insertion method which inserts an operation into the first available idle time interval of its assigned machine is used to make full use of the machine resources. For example, *O*(4, 3) goes after *O*(6, 3) on machine 2 according to the operation sequence in Table 5. But in Fig 4, there is an interval of idle time on machine 2 between the completion time (80) of *O*(4, 2) and the starting time (117) of *O*(6, 3). Meanwhile, the processing time of *O*(4, 3) (32) is smaller than the length of the time interval (117 - 80 = 37). Thus *O*(4, 3) is inserted into this interval and begins at the completion time of *O*(4, 2) (80). The pseudo code of the decoding procedure is shown in Fig 5.

|  |
| --- |
| **Procedure**: Decoding Procedure |
| **Input**: *OS1×n*, the array of operation sequence, n is the total number of operations. |
| **Output**: makespan of a complete schedule |
| **for** *Oji* **in** *OS1×n* **do** |
| *Mji =* the assigned machine for *Oji* |
| search an available idle time interval on machine *Mji* from left to right for operation *Oji* |
| **if** such a time interval is found **then** |
| the operation is inserted there |
| **else** |
| the operation is scheduled at the end of machine *Mji* |
| **end if** |
| **end for**  get the makespan of the schedule |
| **return** the makespan |
| **end procedure** |

Figure 5. Pseudo code of the decoding procedure.

4.2.5 Node selection strategy after *Ns* MCTS simulations

For the sake of simplicity, the start time of an operation obtained by the procedure proposed in Section 5.2.4 is denoted by *SD*, and *tc* denotes the moment when the MCTS completes N simulations. We record the time interval from the rescheduling point to *tc*, and the time interval is called as search time. And Search time of the first ten operations by MCTS is listed Table 2. As we can see from Table 2, after 1.39 seconds from the rescheduling point *tr*, the MCTS completed *N* simulations and the operation *O*(6, 4) was selected, and operation *O*(9, 2) was selected after 2.75 seconds from *tr*.

Table 3

Processing time in the assigned machine and *SD* of each child operation.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Operations | *O*(2, 4) | *O*(9, 2) | *O*(6, 4) | *O*(11, 2) | *O*(1, 4) | *O*(10, 2) |
| Assigned machine | 3 | 4 | 2 | 2 | 3 | 4 |
| Processing time | 10 | 21 | 20 | 5 | 17 | 4 |
| *SD* (s) | 64 | 28 | 39 | 39 | 64 | 28 |



Fig 6. Decoded schedule of the operations in Table 3.

It should be noted that when *Ns* MCTS simulations is completed, we don't always choose the node that has been visited the most, the reasons are as follows. Assume that the operations of one root node’s child nodes are *O*(9, 2), *O*(2, 4), *O*(1, 4) , *O*(6, 4), *O*(11, 2), and *O*(10, 2) respectively, which are called child operations. The predefined processing time of each operation on the assigned machine is listed in Table 3. After N MCTS simulations, we get the number of visited time for each operation, and the corresponding *tc* = 30s. First, we decode all the child operations in descending order of the number of visited time respectively, and get their *SD*, which is also listed in Table 3. Second, for each operation, if *tc* > *SD*, then *tc* will be the earliest start time of the operation, and we call the operation a "delayed operation". The decoding method proposed in Section 5.2.4 will be used to decode the "delayed operation". For example, operation *O*(9, 2) is a "delayed operation" because *tc* (30) > *SD* (28) listed in Table 3. In Fig 6, there is an interval of idle time on machine 4 between *tc* and the starting time of *O*(8, 3) (53). Meanwhile, the processing time of *O*(9, 2) (21) is smaller than the length of the time interval (53 - 28 = 25). Thus *O*(9, 2) is inserted into this interval and begins at *tc*. *O*(9, 2) is also a "delayed operation", which is to be processed on machine 4 at the maximum value of the completion time of *O*(10, 1) and *O*(8, 3). Finally if there is no “delayed operation”, select the node with the most number of visited time and decode its operation into the Gantt chart. We designed the node selection strategy for two reasons: (1) To avoid the “delayed operation” being delayed for too long to be processed, and eventually increase the makespan of the schedule. (2) If the child node represented by the “delayed operation” has not the most number of visited time, but was chosen, this is a way of exploring, which may lead to a complete schedule with a small makespan. Let *O* be a list of child operations of the root node, and *U* be schedule obtained so far. The pseudo code of the node selection strategy is shown in Fig 7.

|  |
| --- |
| **Procedure**: Node selection strategy after N MCTS simulations |
| **Input**: *tc* , *O*, and *U* |
| **Output**: a schedule |
| **for** *Oji* **in** *O* **do** |
| decode *Oji* and get its *SD* |
| **If** *SD* < *tc* **then** |
| Store *Oji* into B // *B is a list for storing delayed operations* |
| **end if** |
| **end for** |
| **if** B is empty **then** |
| select the node with the most number of visited time |
| decode its operation into *U* |
| **else** |
| **for** *Oji* **in** B **do** |
| *tc* is the earliest start time of *Oji* |
| *Mji* is the assigned machine of *Oji* |
| search an available idle time interval on machine *Mji* from left to right for *Oji* |
| **if** such a time interval is found **then** |
| *Oji* is inserted there |
| **else** |
| *Oji* is scheduled at the end of machine *Mji* |
| **end if** |
| **end for** |
| **end if** |
| **return** the schedule |
| **end procedure** |

Fig 7. Pseudo code of the node selection strategy.

4.2.6 Update of the job shop state

Once the rescheduling procedure is triggered, the shop state should be updated at first.

1. At time A, information left from the previous schedule should be collected, which includes the remaining unprocessed operations, and the operations that are being processed on each machine at A. Meanwhile, information about real-time events and the current available machines must also be gathered.
2. Update the available time for all the machines.
3. Update the release time for all the jobs
4. Experimental design and results

5.1 Experimental design

There are no detailed benchmark instances for DJSSP so far. In order to evaluate the performance of the proposed solution methods, dynamic job shop problem instances based on Lawrence (1984) static job shop benchmark problem instances generated in this paper. Each problem takes into account dynamic events such as machine breakdown, new job arrival, order cancellation and change in the processing time of an operation. The generated problem set is classified as small, medium and large-sized problems. Problems which have less than 60 operations are categorized as small size problems (5×5, 6×5, 8×5, 10×5). Problems which have between 60 and 100 operations are categorized as medium size problems (10×6, 15×5, 10×8, 10×9) and problems which have 100 or more operations are categorized as large size problems (10×10, 20×5, 22×5, 12×10, 13×10, 20×7, 15×10). The benchmark instances are given in **(Github Address)**. The proposed approach is coded using Python programming language and tested on a PC with Intel Core i7 3.60 GHz CPU and 8 GB of RAM.

In order to illustrate the potential of the proposed method for DJSS problem, it is compared with some common dispatching rules that widely used in literature (Dominic, Kaliyamoorthy et al. 2004) (Sha and Liu 2005) (Zandieh and Adibi 2010). A list of these rules is as follows:

● SPT (Shortest Processing Time First): Highest priority is given to the waiting operation with the shortest operation time

● LPT (Longest Processing Time First): Highest priority is given to the waiting operation with the longest operation time.

● FIFO (The first in first out): Highest priority is given to the waiting operation that arrived to the queue first.

● LIFO (The last in first out): Highest priority is given to the waiting operation that arrived to the queue last.

●SRPT (Shortest Remaining Processing Time First): Highest priority is given to the waiting operation associated with the job having the least total processing time remaining to be done.

●LRPT (Longest Remaining Processing Time First): Highest priority is given to the waiting operation associated with the job having the most amount of total processing time remaining to be done.

For each problem, the proposed methods are run 5 times and the best results that give a minimum completion time are accepted as the solution.

5.2 Results and discussions

The solutions for the proposed method and dispatching rules are summarized in Table 5, where overall best solutions of each problem instance are highlighted in bold and the deviations of each method related to the best solution are presented. As an example, the data, solution and Gantt chart of a 10×5 dynamic job shop scheduling problem with the proposed method are given in Table 4 and Figure 8 respectively. Figure 9 shows the comparison of the proposed method and dispatching rules in simulated DJSP with different scales. The x-axis represents the scale of DJSP, and the y-axis represents the performance measure - makespan. And the machine utilization is summarized in Table 6, where overall maximum machine utilization of each problem instance are highlighted in bold.

**Table 4**

Solution of 10×5 dynamic job scheduling problem with the proposed method.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Machines | M1 | Job Sequence | J7 | J6 | J1 | J4 | J2 | J10 | J11 | J13 | J9 | J12 | J8 | J10 |  |
|  |  | Start Time | 0 | 53 | 98 | 185 | 244 | 292 | 384 | 429 | 452 | 525 | 540 | 578 |  |
|  |  | Finish Time | 53 | 98 | 185 | 244 | 292 | 364 | 396 | 452 | 500 | 540 | 578 | 596 |  |
|  | M2 | Job Sequence | J1 | J6 | J9 | J11 | J7 | J8 | J13 | J2 | J12 | J4 | J10 |  |  |
|  |  | Start Time | 0 | 119 | 197 | 237 | 263 | 332 | 412 | 429 | 483 | 525 | 626 |  |  |
|  |  | Finish Time | 72 | 197 | 237 | 257 | 309 | 387 | 429 | 483 | 525 | 559 | 646 |  |  |
|  | M3 | Job Sequence | J9 | J8 | J10 | J6 | J1 | J2 | J12 | J11 | J4 | J7 | J13 |  |  |
|  |  | Start Time | 0 | 49 | 136 | 201 | 324 | 390 | 429 | 474 | 561 | 598 | 610 |  |  |
|  |  | Finish Time | 49 | 136 | 201 | 284 | 390 | 429 | 474 | 561 | 598 | 610 | 646 |  |  |
|  | M4 | Job Sequence | J6 | J9 | J2 | J2 | J2 | J7 | J10 | J4 | J8 | J13 | J1 | J11 | J12 |
|  |  | Start Time | 0 | 49 | 132 | 132 | 184 | 192 | 263 | 280 | 299 | 364 | 412 | 561 | 588 |
|  |  | Finish Time | 28 | 132 | 159 | 159 | 192 | 263 | 280 | 299 | 332 | 412 | 472 | 588 | 629 |
|  | M5 | Job Sequence | J8 | J5 | J2 | J6 | J11 | J1 | J12 | J7 | J4 | J13 | J9 | J10 |  |
|  |  | Start Time | 0 | 12 | 35 | 98 | 159 | 224 | 324 | 419 | 448 | 494 | 553 | 596 |  |
|  |  | Finish Time | 12 | 35 | 40 | 103 | 224 | 324 | 419 | 448 | 494 | 553 | 566 | 623 |  |

Figure 8. Gantt chart of the 10×5 dynamic job shop problem with the proposed method.

Table 5 shows that the proposed method provides better solutions in comparison to classical dispatching rule-based approach which is commonly used in dynamic scheduling environments. Deviations of the results from the proposed method, which provides the best solution for each problem size, are also given. According to these values, the proposed provides better results by 0-48.88% when compared to the classical dispatching rules. The deviation of classical dispatching rules from the proposed method increases dramatically as the problem sizes gets bigger. Figure 9 indicates that with the increasing of problem size, the performance measure of SPT, LPT, FIFO, LIFO and SRPT increase largely. However, the performance measure of increases steadier and closely follows the performance curve of the proposed method. Table 6 shows that maximum machine utilization are obtained by the proposed method in comparison to classical dispatching rule-based approaches in all dynamic job shop instances. According to the above analysis, it is easy to conclude that the proposed method provides superior solutions than the most commonly used dispatching rules for all problem instances in terms of both solution quality and machine utilization, especially in large-scale problems. Although the dispatching rules can generate the schedule instantly, our method can generate the schedule timely and quickly while generating a better schedule, and schedule is generated faster than it is utilized. All these results indicate that the combination of dispatching rules and MCTS is an excellent and practical method for rescheduling in dynamic job shop.

Table 5 The makespan of the proposed method and dispatching rules in the DJSP instances.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Problem Size | Methods | |  | |  | |  | |  | |  | |  | |
|  | MCTS-ERM | | SPT | | LPT | | FIFO | | LIFO | | SRPT | | LRPT | |
|  | *Cmax* | *D*(%) | *Cmax* | *D*(%) | *Cmax* | *D*(%) | *Cmax* | *D*(%) | *Cmax* | *D*(%)) | *Cmax* | *D*(%) | *Cmax* | *D*(%) |
| 5×5 | **56** | 0 | 64 | 14.26 | 58 | 3.57 | 64 | 14.29 | 58 | 3.57 | 64 | 14.29 | 58 | 3.57 |
| 6×5 | **544** | 0 | 583 | 7.17 | 669 | 22.98 | 591 | 8.64 | 645 | 18.57 | 598 | 9.93 | 630 | 15.81 |
| 8×5 | **823** | 0 | 868 | 5.47 | 901 | 9.48 | 823 | 0.00 | 1017 | 23.57 | 1071 | 30.13 | 850 | 3.28 |
| 10×5 | **646** | 0 | 733 | 13.47 | 818 | 26.63 | 692 | 7.12 | 678 | 4.95 | 755 | 16.87 | 706 | 9.29 |
| 10×6 | **810** | 0 | 930 | 14.81 | 1069 | 31.98 | 922 | 13.83 | 827 | 2.1 | 1055 | 30.25 | 817 | 0.86 |
| 15×5 | **1065** | 0 | 1316 | 23.57 | 1258 | 18.12 | 1137 | 6.76 | 1072 | 0.66 | 1218 | 14.37 | 1089 | 2.25 |
| 10×8 | **1026** | 0 | 1182 | 15.2 | 1387 | 35.19 | 1217 | 18.62 | 1157 | 12.77 | 1216 | 18.52 | 1208 | 17.74 |
| 10×9 | **1089** | 0 | 1111 | 2.02 | 1397 | 28.28 | 1130 | 3.76 | 1287 | 18.18 | 1321 | 21.3 | 1157 | 6.24 |
| 20×5 | **1367** | 0 | 1484 | 8.56 | 1727 | 26.34 | 1535 | 12.29 | 1605 | 17.41 | 1754 | 28.31 | 1465 | 7.17 |
| 10×10 | **1067** | 0 | 1174 | 10.03 | 1276 | 19.59 | 1253 | 17.43 | 1164 | 9.09 | 1298 | 21.65 | 1115 | 4.50 |
| 12×10 | **1115** | 0 | 1328 | 19.10 | 1660 | 48.88 | 1279 | 14.71 | 1420 | 27.35 | 1625 | 45.74 | 1239 | 11.12 |
| 13×10 | **1074** | 0 | 1181 | 9.96 | 1483 | 38.08 | 1206 | 12.29 | 1417 | 31.94 | 1421 | 32.31 | 1192 | 10.99 |
| 20×7 | **1342** | 0 | 1544 | 15.05 | 1739 | 29.58 | 1566 | 16.69 | 1517 | 13.04 | 1785 | 33.01 | 1369 | 2.01 |
| 15×10 | **1226** | 0 | 1290 | 5.22 | 1567 | 27.81 | 1471 | 19.98 | 1593 | 29.93 | 1399 | 14.11 | 1279 | 4.32 |
| Average | 946.43 |  | 1056.29 |  | 1210.79 |  | 1063.29 |  | 1104.07 |  | 1184.29 |  | 1012.43 |  |
| Deviation % | 0.00 |  | 11.61 |  | 27.93 |  | 12.35 |  | 16.66 |  | 25.13 |  | 6.97 |  |

*D*(%) = (Obtained Average - Best Average)/Best Average

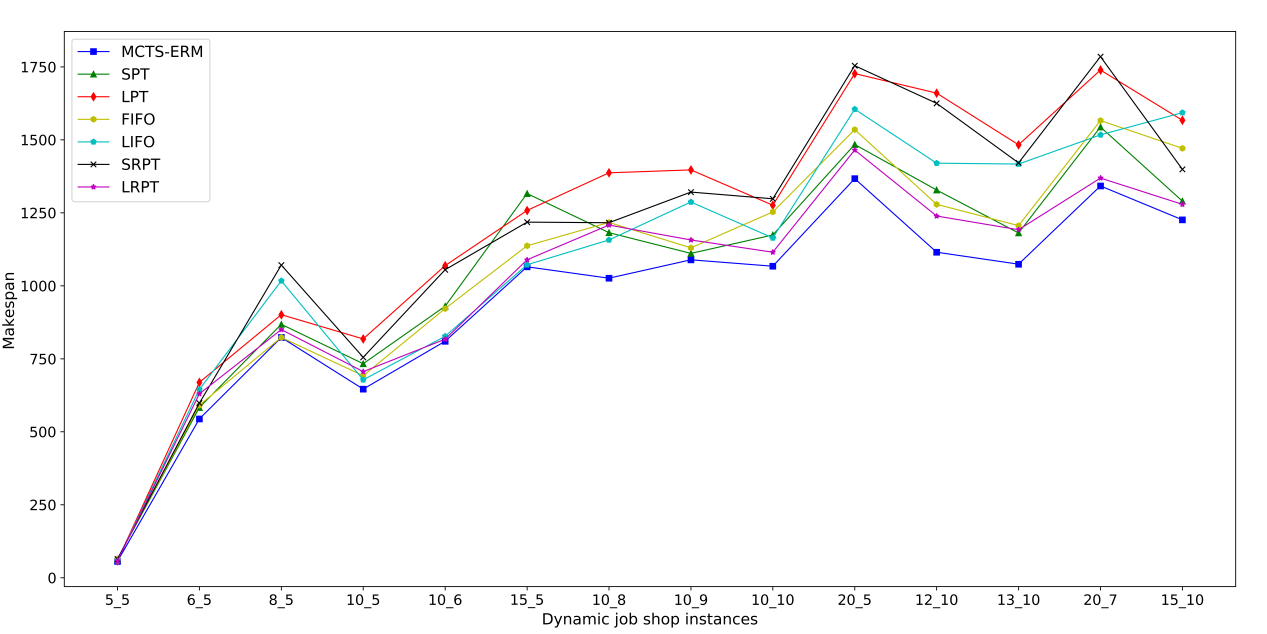


Figure 9. Comparison of the proposed method and dispatching rules.

Table 6 Machine utilization (%) of the proposed method and dispatching rules in the DJSP instances.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Problem Size | MCTS-ERM | SPT | LPT | FIFO | LIFO | SRPT | LRPT |
| 6×5 | **69.10** | 63.64 | 63.92 | 62.45 | 67.36 | 62.45 | 69.07 |
| 6×5 | **73.49** | 70.40 | 64.56 | 68.06 | 62.40 | 69.14 | 65.47 |
| 8×5 | **74.07** | 72.46 | 63.72 | 67.79 | 58.43 | 55.07 | 72.94 |
| 10×5 | **82.08** | 82.01 | 69.10 | 71.82 | 76.78 | 70.21 | 76.11 |
| 10×6 | **81.83** | 72.83 | 63.70 | 72.64 | 76.42 | 65.18 | 78.39 |
| 15×5 | **87.14** | 81.48 | 74.62 | 80.97 | 80.50 | 72.56 | 82.86 |
| 10×8 | **66.95** | 64.94 | 57.44 | 62.54 | 65.54 | 60.50 | 64.71 |
| 10×9 | **72.00** | 71.40 | 56.43 | 68.71 | 62.59 | 58.85 | 69.49 |
| 20×5 | **89.97** | 83.72 | 76.36 | 85.48 | 76.67 | 76.68 | 89.57 |
| 10×10 | **68.58** | 67.35 | 54.63 | 56.91 | 58.16 | 55.15 | 62.18 |
| 12×10 | **69.13** | 65.00 | 48.08 | 61.86 | 56.16 | 51.54 | 65.31 |
| 13×10 | **72.75** | 70.03 | 60.23 | 67.55 | 56.78 | 59.68 | 68.06 |
| 20×7 | **90.36** | 82.29 | 74.01 | 78.77 | 79.90 | 74.08 | 89.46 |
| 15×10 | **72.37** | 70.24 | 57.79 | 63.44 | 62.38 | 64.92 | 72.14 |

1. Conclusion

In this paper, a MCTS-based rescheduling method is proposed to solve the dynamic job shop scheduling problem with several common real-time events, including random job arrivals, machine breakdowns, order cancellation, and change in the processing of an operation. At any rescheduling point, the simulator generates disturbances for next step, then the reschedule is generated quickly by the proposed method. The main contributions of this paper are as follows:

* Most publications in the dynamic job scheduling area only assume that a single real-time event occurs at a rescheduling point, but ignore various combinations of mutiple uncertain events in a real manufacturing system, although it is less likely to occur than the former, more complex and destructive. This paper takes this situation into consideration, and the proposed method achieves a better solution than the scheduling rules for both situations.
* The proposed method in this paper has achieved high-quality solutions to various dynamic job shop conditions, and without any domain knowledge. This indicates that the method has good generalization and robustness, furtherly it can be extended to other types of scheduling problems, such as dynamic flow shop scheduling problems and dynamic flexible job shop scheduling problems.
* Although the proposed method takes a longer required CPU time than the dispatching rules, it can improve the schedule efficiency significantly and guarantee that no downtime is required to wait for the rescheduling scheme to be generated. Otherwise, the required CPU time of the proposed method can be acceptable for real manufacturing systems.

Appendix A. The dynamic job shop instances used in our experiments

In the following tables, event type “0” indicates machine breakdown, event type ‘‘1” indicates new job arrival, event type ‘‘2” indicates change in the processing time of an operation and event type ‘‘3” indicates order cancellation. For instance, in Table 7, dynamic event D5 means machine 1 breakdowns at the 12th second and it continues for 3 s, dynamic event D4 means, a new job J6 arrives at the 5th second and this job’s processed sequence on the machines is M3, M5, M2, M4, M1 and processing times on each machine are 7, 3, 6, 1, 4 respectively, dynamic event D1 means the processing time of job J4 on machine M1 changes as 6 s, and dynamic event D8 means job5’s remaining processing tasks are cancelled at the 23th second.

Asta, S., D. Karapetyan, A. Kheiri, E. Ozcan and A. J. Parkes (2016). "Combining Monte-Carlo and hyper-heuristic methods for the multi-mode resource-constrained multi-project scheduling problem." Information Sciences **373**: 476-498.

Bouzy, B. and B. Helmstetter (2004). Monte-Carlo Go developments. Advances in Computer Games: Many Games, Many Challenges. H. J. VanDenHerik, H. Iida and E. A. Heinz. **135:** 159-174.

Chiang, T.-C. and H.-J. Lin (2012). Flexible Job Shop Scheduling Using a Multiobjective Memetic Algorithm. Advanced Intelligent Computing Theories and Applications: With Aspects of Artificial Intelligence. D. Huang, Y. Gan, P. Gupta and M. M. Gromiha. **6839:** 49-56.

Coulom, R. (2007). Efficient selectivity and backup operators in Monte-Carlo tree search. Computers and Games. H. J. VanDenHerik, P. Ciancarini and H. H. L. Donkers. **4630:** 72-83.

Dominic, P. D. D., S. Kaliyamoorthy and M. S. Kumar (2004). "Efficient dispatching rules for dynamic job shop scheduling." International Journal of Advanced Manufacturing Technology **24**(1-2): 70-75.

Furuoka, R. and S. Matsumoto (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.

Gelly, S. and D. Silver (2011). "Monte-Carlo tree search and rapid action value estimation in computer Go." Artificial Intelligence **175**(11): 1856-1875.

Kashan, A. H., B. Karimi and F. Jolai (2010). "An effective hybrid multi-objective genetic algorithm for bi-criteria scheduling on a single batch processing machine with non-identical job sizes." Engineering Applications of Artificial Intelligence **23**(6): 911-922.

Kocsis, L. and C. Szepesvari (2006). Bandit based Monte-Carlo planning. Machine Learning: Ecml 2006, Proceedings. J. Furnkranz, T. Scheffer and M. Spiliopoulou. **4212:** 282-293.

Lu, C.-L., S.-Y. Chiu, J. Wu and L.-P. Chao (2016). "Dynamic Monte-Carlo Tree Search Algorithm for Multi-Objective Flexible Job-shop Scheduling Problem." Applied Mathematics & Information Sciences **10**(4): 1531-1539.

Nouiri, M., A. Bekrar and D. Trentesaux (2018). "Towards Energy Efficient Scheduling and Rescheduling for Dynamic Flexible Job Shop Problem." Ifac Papersonline **51**(11): 1275-1280.

Pellier, D., B. Bouzy and M. Metivier (2010). An UCT Approach for Anytime Agent-Based Planning. Advances in Practical Applications of Agents and Multiagent Systems. Y. Demazeau, F. Dignum, J. M. Corchado and J. B. Perez. **70:** 211-220.

Rangsaritratsamee, R., W. G. Ferrell and M. B. Kurz (2004). "Dynamic rescheduling that simultaneously considers efficiency and stability." Computers & Industrial Engineering **46**(1): 1-15.

Sha, D. Y. and C. H. Liu (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.

Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel and D. Hassabis (2016). "Mastering the game of Go with deep neural networks and tree search." Nature **529**(7587): 484-+.

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

Vinod, V. and R. Sridharan (2008). "Scheduling a dynamic job shop production system with sequence-dependent setups: An experimental study." Robotics and Computer-Integrated Manufacturing **24**(3): 435-449.

Wu, T.-Y., I. C. Wu, C.-C. Liang and Ieee (2013). Multi-Objective Flexible Job Shop Scheduling Problem Based on Monte-Carlo Tree Search. 2013 Conference on Technologies and Applications of Artificial Intelligence**:** 73-78.

Zandieh, M. and M. A. Adibi (2010). "Dynamic job shop scheduling using variable neighbourhood search." International Journal of Production Research **48**(8): 2449-2458.

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