Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.18576/amis/100431

Dynamic Monte-Carlo Tree Search Algorithm for Multi-Objective Flexible Job-shop Scheduling Problem

Chun-Liang Lu[1], *Shih-Yuan Chiu*[2], *Jiinpo Wu*[3] *and Li-Pen Chao*[3,*]

[1] Department of Information Technology, Ching Kuo Institute of Management and Health, Keelung County, 203, Taiwan.
[2] Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien County, 974, Taiwan.
[3] Department of Information Management, Tamkang University, New Taipei City, 251, Taiwan.

Received: 15 Sep. 2014, Revised: 1 Mar. 2015, Accepted: 21 Sep. 2015
Published online: 1 Jul. 2016

Abstract: Scheduling is a key factor in real-world production management and manufacturing system. The Job-shop Scheduling Problem (JSP) is important in combinatorial optimization and well known as an NP-hard. The Flexible Job-shop scheduling problem (FJSP) is an extension of the JSP, which allows an operation may be processed by any machine from a given set. It is more complex than JSP and quite difficult to achieve an optimal solution with traditional optimization approaches owing to the high computational complexity. The intent of this paper is to develop an efficient Dynamic Monte-Carlo Tree Search model for solving the multi-objective FJSP. First, a Sequential Operation-Machine Assignment (SOMA) scheme is proposed for encoding representation, lead to potentially produce feasible candidate solutions for the FJSP. Then evolution-based FJSP mapping to a general tree search structure via the SOMA is successively completed. Second, the original single-objective UCT (Upper Confidence bound apply to Tree) algorithm is modified (called NSUCT) by using a Non-dominated Sorting strategy to be able to deal with multi-objective optimization problems. Third, the dynamic Monte-Carlo sampling technique is adopted for the tree search evaluation and guided with NSUCT to balance between exploitation and exploration during the evolution process. Finally, the Multi-Objective Monte-Carlo Tree Search (MOMCTS) algorithm is proposed to solve the FJSP for finding the Pareto-optimal solutions. Several popular benchmark problems with various conditions are considered to compare the proposed MOMCTS algorithm with the published algorithms. Simulation results show that the MOMCTS is able to acquire wide range of Pareto-optimal solutions. In addition, the more decision-makings of the results in the form of Gantt chart under the same Pareto-optimal solutions can be obtained.

Keywords: Flexible Job-shop Scheduling Problem, Multi-Objective Monte-Carlo Tree Search, Pareto-optimal solutions.

1 Introduction

The Job-shop Scheduling Problem (JSP) is important in combinatorial optimization and well known as an NP-hard [1], which concerned with allocating limited resources to tasks to optimize some performance criterion. The classical JSP consists of jobs and machines. Each job must be performed on each machine in a pre-defined sequence, and each operation of a job is to be processed only one machine at a time. The Flexible Job-shop scheduling problem (FJSP) [2] is an extension of the JSP, which allows an operation may be processed by any machine from a given set. It is more complex than JSP because of the two decisions has to be made: assignment of operations to machines and sequencing the operations on each machine. It is quite difficult to achieve an optimal solution with traditional optimization approaches owing to the high computational complexity.

In the literature, different optimization approaches have been proposed to solve the FJSP, including genetic algorithm (GA) [3,4], simulated annealing (SA) [5], ant colony optimization (ACO) [6], or particle swarm optimization (PSO) [7,8], have made promising results. The combination of evolutionary algorithm and local search approaches for solving the FJSP have been increasingly investigated by researchers and achieved considerable good solutions. Xia and Wu [9] proposed the PSO-SA method for solving FJSP on three public benchmark datasets represented by problem (problem 8×8, 10×10 and 10×15) and provided solutions with good quality to show the effectiveness of this approach. Ho and Tay [10] presented the multi-objective

* Corresponding author e-mail: lbchao@ems.cku.edu.tw

evolutionary algorithm with guided local search (MOEA-GLS) algorithm to solve FJSP and obtained better Pareto-optimal solutions than PSO-SA method on the same three benchmark problems. However, the authors in [10] did not deal with the decision-makings of different types in Gantt chart under the same Pareto-optimal solutions.

In this paper, the Multi-Objective Monte-Carlo Tree Search (MOMCTS) algorithm is proposed to solve the FJSP for finding the Pareto-optimal solutions. Several popular benchmark problems with various conditions are considered to compare the proposed MOMCTS algorithm with the published methods. Simulation results show that the MOMCTS can achieve wide range of Pareto-optimal solutions. In addition, the more decision-makings of Gantt chart with the same Pareto-optimal solutions can be obtained. The organization of the rest paper is as follows: Section 2 introduces the FJSP formulations. Section 3 illustrates the MCTS framework and UCT algorithm. In Section 4, SOMA scheme, problem transformation, external repository and modification of UCT, Path Random Search (PRS) method, and the proposed MOMCTS algorithm are presented. Several public benchmark problems are used to validate the proposed method and comparison results are shown in Section 5. Finally, conclusions are made in Section 6.

2 FJSP Description

The FJSP is a manufacturing strategy to properly organize a set of machines to process a set of jobs for some objectives. The FJSP is generally described as follows: The problem is to organize the execution of n jobs $J_i(i = 1,2,...,n)$on m machines$M_k(k = 1,2,...,m)$, where each job J_i needs o_i operations on the order of restraint and each working procedure of job can be processed by multiple processes of M machines. The total number of operations in all jobs isO_t, $O_t = \sum_{i=1}^{n} o_i$. M_{ij}means the collection of the useable machines about j-th operation of i-th job, $M_{ij} \in \{1,2,...,m\}$, $O_{i,j,k}$ means the j-th operation of the i-th job can use k-th machine, $p_{i,j,k}$means the required time of j-th operation of i-th job processed on k-th machine, $1 \le i \le n$, $1 \le j \le o_i$, $1 \le k \le m$. In the FJSP, the following assumptions are usually made [2]. The task is to find a set of solutions and three criteria of FJSP are considered and described as follows:

(a) The total workload of all machines

$$F_1 = \sum p_{i,j,k} \tag{1}$$

where $p_{i,j,k}$ denotes the processing time of j-th operation of i-th job in k-th machine.

(b) The workload of the critical machine

$$F_2 = \max\{W_1, W_2, ..., W_m\} \tag{2}$$

where W_k denotes the workload of k-th machine M_k.

(c) The completion time of the critical job

$$F_3 = \max\{CT_1, CT_2, ..., CT_n\} \tag{3}$$

where CT_i denotes the completion time of i-th jobJ_i.

3 Structure of Monte-Carlo Tree Search (MCTS)

3.1 Monte-Carlo Tree Search Framework

Monte-Carlo Tree Search (MCTS) [11] is a best-first search method which uses Monte-Carlo sampling as its evaluation function and is both explorative and exploitative. Each node of the search tree contains a current problem state, the possible actions, and some other information about this state. Using the results of previous explorations, the algorithm gradually grows a random search tree in memory, and successively becomes better at accurately estimating the values of the most promising moves. If there is time left, all sampling search and evaluation strategic phases will be run again. MCTS can be stopped anytime, the more time the program runs, the stronger the program may execute.

3.2 The UCT Algorithm

The algorithm UCT is the extension of UCB1 [12] to min-max tree search. It has its origins in solving the multi-armed bandit problem. A K-armed bandit problem [13] is described to find the strategy for a bandit to choose its K arms to get the maximum reward. The problem is defined by the sequence of random variables$X_{i,n}$, $i = 1,2,...,K$, $n \ge 1$, where i is the index of the gambling machine and n is the number of times the machine was played. The UCB1 policy was originally given in [14], which ensures the optimal machine is played exponentially more often than any other machine uniformly when the rewards are in [0, 1]. The essence is choosing the machine i which maximizes the following formula:

$$UCT_i = \bar{x_i} + C_e \cdot \sqrt{\frac{\ln n}{n_i}} \tag{4}$$

where $\bar{x_i}$is the mean reward obtained from machine i so far, n_i is the number of times machine i has been played and n is the overall number of plays done. The const C_e must be adjusted to balance between exploitation and exploration. For the tree search procedure, the idea of UCT is to consider each node as an independent bandit, with its child nodes as independent arms. From the current search position, the UCT method is to select action according to knowledge contained within the search tree. Each state in the tree estimates its UCT value for each node by Monte-Carlo sampling simulation. As

more simulation results propagate up the tree, the search policy improves at every actual time-step, and the Monte-Carlo is based on more accurate returns.

4 Methods

4.1 Representation and SOMA scheme

The evolution-based heuristic method can be achieved better efficiency by means of the appropriate encoding representation. Chen et al. [15] originally uses an A-B string representation. A string indicates a list of all operations of all jobs while B string denotes a list of operations that are processed on each machine. But the repair mechanism is required to deal with a deadlock situation. This study extended the feature to generate a {operation-machine} hybrid label as encoding representation, and the Sequential Operation Machine Assignment (SOMA) scheme has been proposed to always produce feasible solutions. Therefore, a repair mechanism such as local search method, to maintain feasibility is not required.

To explain this scheme, the example (3 jobs, 2 machines, and 7 operations) is considered (in Table 4.1) to detail the encoding representation and SOMA scheme step by step. For instance, operation $O_{1,1}$ of job 1 can be processed on machine M_a for 4 units of time, then to machine M_b for 3 units of time, and so on. Here, we simplify the machine selection with the least processing units of time. The symbols of candidate and selected {operation-machine} pairs are indicated by '△' and '◯', respectively.

Step 1: From Table 4.2, we can generate the {operation-machine}candidate set C_1= {$O_{1,1}$-M_a,$O_{1,1}$-M_b, $O_{2,1}$-M_a,$O_{2,1}$-M_b,$O_{3,1}$-M_a,$O_{3,1}$-M_b}. In our simplified case, we choose the {operation-machine} pair with the least processing units of time fromC_1 for computing each processing time {4, 3, 1, 3, 6, 7}, to select the {$O_{2,1}$-M_a} with the minimum value 1 as the selection set denoted by S_1= {(2.1a)} (in Table 4.3).

Step 2: From Table 4.3, the candidate operation setO_2={$O_{1,1}$,$O_{2,2}$,$O_{3,1}$}, then to generate the candidate set C_2= {$O_{1,1}$-M_a,$O_{1,1}$-M_b, $O_{2,2}$-M_a,$O_{2,2}$-M_b,$O_{3,1}$-M_a,$O_{3,1}$-M_b}. Compute each processing time {4, 3, 7, 5, 6, 7} fromC_2, to select the {$O_{1,1}$-M_b} with the minimum value 3, and the selection set S_2={(2.1a), (1.1b)} (in Table 4.4).

Step 3: From Table 4.4, the candidate operation setO_3={$O_{1,2}$,$O_{2,2}$,$O_{3,1}$ }, then to generate the candidate set C_3= {$O_{1,2}$-M_a,$O_{1,2}$-M_b, $O_{2,2}$-M_a,$O_{2,2}$-M_b,$O_{3,1}$-M_a,$O_{3,1}$-M_b}. Compute each processing time {7, 10, 7, 5, 6, 7} fromC_3, to select the {$O_{2,2}$-M_b} with the minimum value 5, and the selection set S_3= {(2.1a), (1.1b), (2.2b)} (in Table 4.5).

Step 4: From Table 4.5, the candidate operation setO_4={$O_{1,2}$, $O_{3,1}$ }, then to generate the candidate set C_4= {$O_{1,2}$-M_a,$O_{1,2}$-M_b, $O_{3,1}$-M_a,$O_{3,1}$-M_b}. Compute each processing time {7, 10, 6, 7} fromC_4, to select the {$O_{3,1}$-M_a} with the minimum value 6, and the selection set S_4= {(2.1a), (1.1b), (2.2b), (3.1a)} (in Table 4.6).

Step 5: From Table 4.6, the candidate operation setO_5={$O_{1,2}$, $O_{3,2}$ }, then to generate the candidate set C_5= {$O_{1,2}$-M_a,$O_{1,2}$-M_b, $O_{3,2}$-M_a,$O_{3,2}$-M_b}. Compute each processing time {7, 10, 6, 5} fromC_5, to select the {$O_{3,2}$-M_b} with the minimum value 5, and the selection set S_5= {(2.1a), (1.1b), (2.2b), (3.1a), (3.2b)} (in Table 4.7).

Step 6: From Table 4.7, the candidate operation setO_6={$O_{1,2}$}, then to generate the candidate set C_6= {$O_{1,2}$-M_a,$O_{1,2}$-M_b}. Compute each processing time {7, 10} fromC_6, to select the {$O_{1,2}$-M_a} with the minimum value 7, and the selection set S_6= {(2.1a), (1.1b), (2.2b), (3.1a), (3.2b), (1.2a)} (in Table 4.8).

Step 7: From Table 4.8, the candidate operation setO_7={ $O_{1,3}$}, then to generate the candidate set C_7= {$O_{1,3}$-M_a,$O_{1,3}$-M_b}. Compute each processing time {5, 4} fromC_7, to select the {$O_{1,3}$-M_a} with the minimum value 4, and the selection set S_7= {(2.1a), (1.1b), (2.2b), (3.1a), (3.2b), (1.2a), (1.3a)} (in Table 4.9).

As shown in Table 4.9, the operations of all jobs are assigned to each machine done and one of the feasible solutions S_7 is obtained. Its Gantt chart is shown in Figure 4.1. The precedence constraints are {(1.1b) before (1.2a) before (1.3b)} (job 1), {(2.1a) before (2.2b)} (job 2), {(3.1a) before {(3.2b)} (job 3), and the operation order should be preserved by the different machine selection. It is obvious that each scheduling solution from any encoding representation based on {operation-machine} and decoding by SOMA should be feasible. Moreover, every encoding representation pair is initialized randomly, so the proposed SOMA based scheme can produce more different scheduling solutions.

4.2 Problem Transformation

Monte-Carlo search proved to be competitive in deterministic algorithm with large stochastically branching factors. As discussed, the effect of the Monte-Carlo Tree Search structure is to efficiently apply solving for the FJSP. Here, the definition of a tree data structure is introduced, and the mapping procedures of the example in Section 4.1 to tree search topology are described as follows.

Assume that a simple data structure (a level tree) with root node r. Every node in the tree is associated with a candidate {operation-machine} hybrid label pair. Let $S_k = \{s_1, s_2, ..., s_k\}, k \geq 1$ be an ordered items set from the selection stage, then the item-set tree T is level k. The representation of each node denoted by {$x.yz$} string symbol for the combination of {operation-machine} with {$O_{x,y}M_z$}. Every node u other than the root node r has a unique parent node. Every non-leaf nodevin T has $d \geq 1$

child nodes. For two nodes u and its child node $v$in T, we will use $\pi(u,v)$ to denote the unique directed path in T which connects u and v. For detailed description, the mapping procedure according to the SOMA approach illustrated in Section 4.1 gives a step by step discussion properly.

Step 1: Initialize to start from the tree T with root node atL_0, then to create its child nodes set N_1= {(1.1a), (1.1b), (2.1a), (2.1b), (3.1a), (3.1b)} at L_1 corresponding to the candidate set items fromC_1 . The construction of search tree topology is shown in Figure 4.2.

Step 2: The search path π_1= {(2.1a)} is determined by the selection setS_1. Start from the search point node (2.1a) at present, to construct its child nodes set N_2= {(1.1a), (1.1b), (2.2a), (2.2b), (3.1a), (3.1b)} at L_2 corresponding to the candidate set C_2.

Step 3:The search path π_2= {(2.1a), (1.1b)} is determined by the selection setS_2. Start from the search point node (1.1b) at present, to construct its child nodes set N_3= {(1.2a), (1.2b), (2.2a), (2.2b), (3.1a), (3.1b)} at L_3 corresponding to the candidate set C_3.

Step 4:The search path π_3= {(2.1a), (1.1b), (2.2b)} is determined by the selection setS_3. Start from the search point node (2.2b) at present, to construct its child nodes set N_4= {(1.2a), (1.2b), (3.1a), (3.1b)} at L_4 corresponding to the candidate set C_4.

Step 5: The search path π_4= {(2.1a), (1.1b), (2.2b), (3.1a)} is determined by the selection setS_4. Start from the search point node (3.1a) at present, to construct its child nodes set N_5= {(1.2a), (1.2b), (3.2a), (3.2b)} at L_5 corresponding to the candidate set C_5.

Step 6:The search path π_5= {(2.1a), (1.1b), (2.2b), (3.1a), (3.2b)} is determined by the selection setS_5. Start from the search point node (3.2b) at present, to construct its child nodes set N_6= {(1.2a), (1.2b)} at L_6 corresponding to the candidate set C_6.

Step 7: The search path π_6= {(2.1a), (1.1b), (2.2b), (3.1a), (3.2b), (1.2a)} is determined by the selection setS_6. Start from the search point node (1.2a) at present, to construct its child nodes set N_7= {(1.3a), (1.3b)} at L_7 corresponding to the candidate set C_7.

Step 8: In Figure 4.3, the search path π_7= {(2.1a), (1.1b), (2.2b), (3.1a), (3.2b), (1.2a), (1.3b)} is determined by the selection setS_7 and the growth of tree search topology procedure ends at level 7.

Throughout the execution of the tree search growth, a node u has bold rim mark if the best search node is chosen with least processing units of time. It is clear that the appropriate choice of the search path π_7 in T is equivalent to one of the feasible solutions S_7 in FJSP. As shown in Figure 4.3, we can successively convert the original evolution-based framework to the randomized tree search model via the SOMA scheme.

4.3 External REP and Modification of UCT

As mentioned in Section 3.2, the original UCT is widely used to deal with uncertainty in a smooth way. For every search node in a single objective optimizations problem, the estimated value is the summation of mean of the value for each child and weighted by the frequency of visits. The UCT selects most of the time the maximum estimated value child nodes if one child-node has a much higher value than the others. However, due to the multi-criteria nature of multiple objectives problems, the optimality of a solution has to be redefined, giving rise to the concept of Pareto-optimal solution. In [16], the authors adopt an external repository (REP) to keep the historical record of the non-dominated solutions found along the search space. The basic concept of REP function is to determine whether a certain solution should be added or not.

Here, the non-dominated sorting strategy in [17] was adopted to modify the formal UCT, named NSUCT (Non-dominated Sorting UCT), to make it suitable for the features of multi-objective problem on the MCTS sampling framework.

$$NSUCT_i = (w_i/t_i) + C_e \cdot \sqrt{\ln t_n / t_i} \tag{5}$$

$$w_i = 1 - (D_i/N_{REP}) \tag{6}$$

where t_i is the number of node v_i has been visited and t_n is the overall number of the parent node of v_i visited done. w_i denoted the win score value for each nodev_i, $D_i$is the number of dominated count among node$v_i$and non-dominated solutions in REP, and$N_{REP}$is the number of non-dominated solutions in REP .

4.4 Path Random Search (PRS) algorithm

Consider to build a tree with its nodes labeled by {operation-machine} symbol in large search state space, the Path Random Search (PRS) algorithm based on the Monte-Carlo sampling technique has been proposed to simulate a random search for estimating the state-action value. For reading convenience, the symbols employed in the sequel of the procedure statement are summarized in Table 4.10. The proposed PRS algorithm is described as follows.

PRS algorithm procedure:

Step 1: Set WS is the working set for current nodes.

Step 2: Use roulette wheel selection to choose one node v from WS according to the probability of processing time for each operation on the machine. The total fitness (TF_p) of the WS from PRS procedure is given by Eq. (7). The probability (p_i) of a selection for each node v_i is Eq. (8). The cumulative probability for each node v_i is Eq. (9). Each time a node v_i for a new choice is

selected a random number r is generated in the range [0, 1]. If $r < q_1$ then select the first node v_1, otherwise select the i-th node v_i such that $q_{i-1} < r \leq q_i$.

$$TF_p = \sum_{i=1}^{N_{ws}} EF_i \ , \ EF_i = (\frac{1}{t_i}) \tag{7}$$

$$p_i = \frac{EF_i}{TF_p} \tag{8}$$

$$q_i = \sum_{j=1}^{i} p_j \tag{9}$$

Step 3: Execute a random search R for the selected node v from Step 2.

Step 4: Obtain path $\pi(v)$ from the root node through node v to bottom node in T.

Step 5: Calculate the fitness value $F(v)$ and Gantt chart $G(v)$ according to$\pi(v)$.

Step 6: Check if the $F(v)$ dominates or non-dominates the solutions in REP, then update and store the fitness value $F(v)$ to REP; otherwise go to Step 8.

Step 7: Check if the $G(v)$is the same as solutions in REP, then store the Gantt chart $G(v)$ to REP; otherwise go to Step 8.

Step 8: Calculate the dominated times with respect to $F(v)$and solutions in REP to obtain the win score value for node v.

Step 9: Calculate the NSUCT value for node v.

Step 10: Update the NSUCT value with selected parent node from v along to root.

Step 11: Return results to main program.

4.5 The proposed MOMCTS approach

For the standard FJSP, the size of search space grows exponentially. It is computationally infeasible to try every possible solution in the whole tree. Instead of dealing with each node once iteratively, we will adopt the Monte-Carlo simulation technique to each state in the tree search procedure within limited time. Once the search tree has been constructed when FJSP complexity increases, the resulting tree tends to be large. Therefore, the dynamic pruning method is adopted to reduce tree sizes for preventing the growth of those branches seems not to improve the predictive result. The proposed MOMCTS algorithm and the procedure statement are detailed as follows:

MOMCTS algorithm procedure:

Step 1: Initialize the parameters including: Monte-Carlo sampling times (N_{mc}) of iteration, the coefficient (C_e) of NSUCT, the number of preserved nodes N_p after pruning at each level, the tree height h. Set $k = 1$, $wk = 1$, wn=empty, WS=empty.

tep 2: Generate all possible candidate child nodes for root node to WS.

Step 3: Execute one random search R for each node in WS.

Step 4: Calculate the fitness value $F(v_i)$ for each node v_i in WS, where$1 \leq i \leq N_{ws}$.

Step 5: Calculate the dominated counter D_i value of nodev_i with other nodes in WS.

Step 6: Calculate the win score value and obtain the NSUCT value.

Step 7: Set $Sim = 0$ and reset the number index of Monte-Carlo simulation iteration.

Step 8: Set $k = wk$. Use roulette wheel selection to choose one node wn from WS according to the probability of NSUCT value for each node in WS at level k. The total fitness (TF_M) of the WS from MOMCTS procedure is given by Eq. (10). The probability (p_i) of a selection for each node wn_i is Eq. (11). The cumulative probability for each node wn_i is Eq. (12). Each time a node wn_i for a new choice is selected a random number r is generated in the range [0, 1]. If $r < q_1$ then select the first node wn_1, otherwise select the i-th node wn_i such that $q_{i-1} < r \leq q_i$.

$$TF_M = \sum_{i=1}^{N_{ws}} NSUCT_i \tag{10}$$

$$p_i = \frac{NSUCT_i}{TF_M} \tag{11}$$

$$q_i = \sum_{j=1}^{i} p_j \tag{12}$$

Step 9: Check if the child nodes of wn are empty, then set $k = k+1$. Generate all possible candidate child nodes for wn at level k. Set $WS = childs(wn)$. Go to Step 12. Otherwise, set $k = k+1$. Let $WS = childs(wn)$.

Step 10: Check if the nodes in WS are not all simulated, then put these not simulated nodes to WS, go to Step 12. Otherwise, next step.

Step 11: Check if $k < h$, then use roulette wheel selection to choose one node wn from WS according to the probability of NSUCT value for each node in WS at level k, go to Step 9. Otherwise, the path search converges and program ends.

Step 12: Execute the **PRS** algorithm procedure.

Step 13: Set $Sim = Sim + 1$. If $Sim \leq N_{mc}$, then go to Step 8. Otherwise, next step.

Step 14: Set $k = wk$. Rank every node at level k by NSUCT value form high to Low. Preserve $N_p$nodes according their ranks with higher value. The other nodes with lower NSUCT value are to be pruned.

Step 15: Set $wk = wk + 1$. Check if $wk < h$, then set $k = wk$ and go to Step 7. Otherwise, program ends.

Table 4.1 Example of the FJSP with 3 jobs, 2 machines, and 7 operations.

		Ma	Mb
J1	O1.1	4	3
	O1.2	7	10
	O1.3	5	4
J2	O2.1	1	3
	O2.2	7	5
J3	O3.1	6	7
	O3.2	6	5

Table 4.2 Initial candidate set C_1 for SOMA.

		Machine	
J1	O1.1	a	b
	O1.2	a	b
	O1.3	a	b
J2	O2.1	a	b
	O2.2	a	b
J3	O3.1	a	b
	O3.2	a	b

Table 4.3 The selection S_1 and candidate C_2.

		Machine	
J1	O1.1	a	b
	O1.2	a	b
	O1.3	a	b
J2	O2.1	a	b
	O2.2	a	b
J3	O3.1	a	b
	O3.2	a	b

Table 4.4 The selection S_2 and candidate C_3.

		Machine	
J1	O1.1	a	b
	O1.2	a	b
	O1.3	a	b
J2	O2.1	a	b
	O2.2	a	b
J3	O3.1	a	b
	O3.2	a	b

Table 4.5 The selection S_3 and candidate C_4.

		Machine	
J1	O1.1	a	b
	O1.2	a	b
	O1.3	a	b
J2	O2.1	a	b
	O2.2	a	b
J3	O3.1	a	b
	O3.2	a	b

Table 4.6 The selection S_4 and candidate C_5.

		Machine	
J1	O1.1	a	b
	O1.2	a	b
	O1.3	a	b
J2	O2.1	a	b
	O2.2	a	b
J3	O3.1	a	b
	O3.2	a	b

Table 4.7 The selection S_5 and candidate C_6.

		Machine	
J1	O1.1	a	b
	O1.2	a	b
	O1.3	a	b
J2	O2.1	a	b
	O2.2	a	b
J3	O3.1	a	b
	O3.2	a	b

Table 4.8 The selection S_6 and candidate C_7.

		Machine	
J1	O1.1	a	b
	O1.2	a	b
	O1.3	a	b
J2	O2.1	a	b
	O2.2	a	b
J3	O3.1	a	b
	O3.2	a	b

Table 4.9 The selection set S_7.

		Machine	
J1	O1.1	a	b
	O1.2	a	b
	O1.3	a	b
J2	O2.1	a	b
	O2.2	a	b
J3	O3.1	a	b
	O3.2	a	b

Table 4.10. Symbols used in the algorithm procedure statement.

Symbol	Description
T	A general search tree
h	Number of tree height
O_i	Number of operations in all jobs ($O_i = h$)
N_{mc}	Number of Monte-Carlo simulation times for each iteration at level k
N_p	Number of preserved nodes after pruning at level k. N_p is the summation of the number of maximum NSUCT node and the number of nodes in REP.
k	Current index number of tree level ($1 \le k \le h$)
wk	Working index number of tree level ($1 \le wk \le h$)
Sim	Current index number of Monte-Carlo simulation iteration ($1 \le Sim \le N_{mc}$)
v	Selected node at level k
WS	Working set for current nodes at level k
N_{ws}	Number of nodes in WS
wn	Working node at level k
childs(wn)	child nodes of working node wn
t	Processing time unit of node v corresponding to its {operation-machine}.
$\pi(v)$	Search path from root through node v to reach bottom node in tree T
$F(v)$	The fitness value according to $\pi(v)$
$G(v)$	The Gantt chart according to $\pi(v)$
R	Execute the random Monte-Carlo simulation search one time
REP	Non-dominate solutions in the external repository

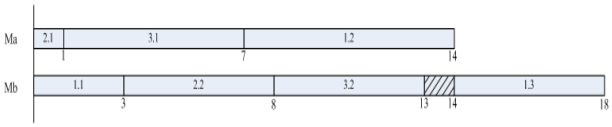

Figure 4.1 The Gantt chart in Step 7 from SOMA scheme.

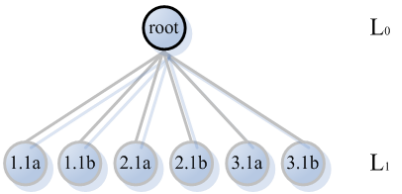

Fig. 4.2 Transformation to tree search structure in Step 1 from SOMA scheme.

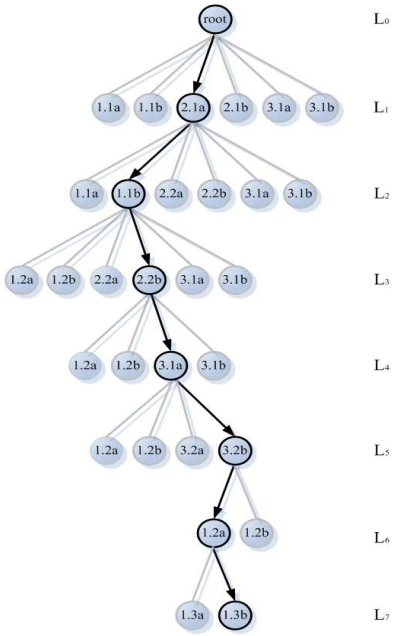

Fig. 4.3 Transformation to tree search structure in Step 8 from SOMA scheme.

5 Simulation Results

Comparing the proposed MOMCTS with the PSO-SA [9] and the MOEA-GLS [10], the computational experiments for several datasets such as three popular benchmarks (problem 8×8, 10×10 and 15×10) and the three problem instances with release dates (problem 4×5, 10×7 and 15×10) are considered. For each problem, the obtained results are reported in table contains three objectives: F_1(total workload), F_2(critical machine workload), F_3(makespan), are mentioned in Section 2. The comparison results for the problem 8×8 (8 jobs, 8 machines, 27 operations), problem 10×10 (10 jobs, 10 machines, 30 operations) , problem 15×10 (15 jobs, 10 machines, 56 operations), problem 4×5 with release date $r_1 = 3, r_2 = 5, r_3 = 1, r_4 = 6$ (4 jobs, 5 machines, 12 operations), problem 10×7 with release date $r_1 = 2, r_2 = 4, r_3 = 9, r_4 = 6$ $r_5 = 7, r_6 = 5, r_7 = 7, r_8 = 4, r_9 = 1, r_{10} = 0$(10 jobs, 7 machines, 29 operations), problem 15×10 with release date $r_1 = 5, r_2 = 3, r_3 = 6, r_4 = 4, r_5 = 9, r_6 = 7, r_7 = 1, r_8 = 2, r_9 = 9, r_{10} = 0$ $r_{11} = 14, r_{12} = 13, r_{13} = 11, r_{14} = 12, r_{15} = 5$ (15 jobs, 10 machines, 56 operations) are presented in Table 5.1 to Table 5.6. The column labeled 'Gantt chart variety' is to display the number of diversity. The symbol 'x' indicates that the authors did not provide the Gantt chart.

For the three benchmarks (problem 8×8, 10×10 and 15×10), to compare the proposed MOMCTS with the PSO-SA method [4], two new non-dominated solutions (77, 12, 14) and (77, 11, 16) can be obtained by the MOMCTS in the problem 8×8. For the problem 10×10, two new solutions (43, 5, 7) and (42, 6, 7) of MOMCTS dominate the PSO-SA and two non-dominated solutions (41, 7, 8) and (42, 5, 8) can be achieved. In the problem 10×15, one new solution (91, 11, 11) of MOMCTS dominates the PSO-SA and one non-dominated solution (93, 10, 11) can be obtained. Although the solutions of MOMCTS are the same as the MOEA-GLS [5], we can see that more diversity of the Gantt charts from MOMCTS can be obtained.

For the other three problem instances with release dates (problem 4×5, problem 10×7 and problem 15×10), to compare the proposed MOMCTS with the AL-CGA method [4], two new non-dominated solutions (32, 8, 16) and (33, 7, 16) can be obtained by the MOMCTS in the problem 4×5. For the problem 10×7, the new solutions (62, 10, 15) of MOMCTS dominate the AL-CGA (63, 10, 18) and (64, 10, 17) solutions. In the problem 10×15, two solutions (91, 11, 23) and (93, 10, 23) from MOMCTS dominates the AL-CGA solutions (91, 11, 24) and (95, 11, 23). Although the solutions of MOMCTS are the same as the MOEA-GLS [5], the more diversity of the Gantt charts from MOMCTS also can be obtained. For example, two different Gantt charts of the solution (93, 10, 23) for the problem 10×15 with release date are shown in Figure 5.1 and Figure 5.2.

Table 5.1 Comparison of results on problem 8×8.

Algorithm	Wtd	Max(Wk)	Makespan	Gantt charts
	F1	F2	F3	variety
PSO-SA	75	12	15	1
	73	13	16	1
MOEA-GLS	75	12	15	x
	73	13	16	x
	77	12	14	x
	77	11	16	x
Our proposed MODMCTS	75	12	15	10
	73	13	16	15
	77	12	14	3
	77	11	16	3

Table 5.2 Comparison of results on problem 10×10.

Algorithm	Wtd	Max(Wk)	Makespan	Gantt charts
	F1	F2	F3	variety
PSO-SA	44	6	7	1
MOEA-GLS	41	7	8	x
	42	5	8	x
	43	5	7	x
	42	6	7	x
Our proposed MODMCTS	41	7	8	9
	42	5	8	13
	43	5	7	7
	42	6	7	6

Table 5.3 Comparison of results on problem 15×10.

Algorithm	Wtd	Max(Wk)	Makespan	Gantt charts
	F1	F2	F3	variety
PSO-SA	91	11	12	1
MOEA-GLS	91	11	11	x
	93	10	11	x
Our proposed MODMCTS	91	11	11	9
	93	10	11	6

Table 5.4 Comparison of results on problem 4×5 with release date.

Algorithm	Wtd	Max(Wk)	Makespan	Gantt charts
	F1	F2	F3	variety
AL-CGA	32	8	18	x
	33	7	18	x
	34	10	16	x
	35	9	16	x
MOEA-GLS	32	8	16	x
	33	7	16	x
Our proposed MODMCTS	32	8	16	5
	33	7	16	2

Table 5.5 Comparison of results on problem 10×7 with release date.

Algorithm	Wtd	Max(Wk)	Makespan	Gantt charts
	F1	F2	F3	variety
AL-CGA	60	12	16	x
	61	11	15	x
	63	10	18	x
	64	10	17	x
	66	10	16	x
MOEA-GLS	60	12	16	x
	61	11	15	x
	62	10	15	x
Our proposed MODMCTS	60	12	16	11
	61	11	15	9
	62	10	15	16

Table 5.6 Comparison of results on problem 15×10 with release date.

Algorithm	Wtd	Max(Wk)	Makespan	Gantt charts
	F1	F2	F3	variety
AL-CGA	91	11	24	x
	95	11	23	x
MOEA-GLS	91	11	23	x
	93	10	23	x
Our proposed MODMCTS	91	11	23	13
	93	10	23	9

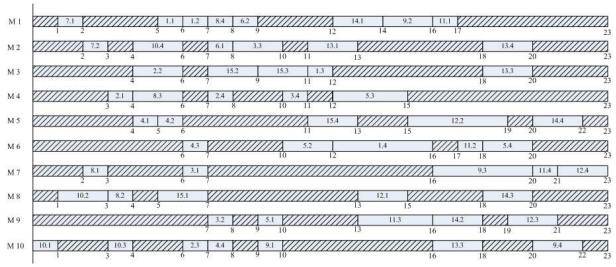

Fig. 5.1. The Gantt chart 1 of (93, 10, 23) for the problem 10×15 with release date.

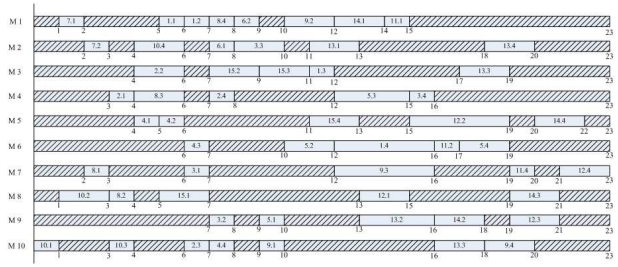

Fig. 5.2. The Gantt chart 2 of (93, 10, 23) for the problem 10×15 with release date.

6 Conclusions

The proposed SOMA scheme for encoding representation is used to always produce feasible solutions. Therefore, the evolution-based FJSP mapping to a general tree search structure via SOMA is successively completed. Next, the modification of formal UCT by the non-dominated sorting strategy, called NSUCT, makes it suitable for the features of multi-objective problems. The proposed MOMCTS approach solving FJSP is compared with the integrated multi-objective approach based on evolutionary method for several popular benchmarks. The computational results validate the effectiveness of the proposed MOMCTS approach, and the more decision-makings under the same Pareto-optimal solutions condition can be obtained.

References

[1] M.R. Garey, D.S. Johnson, and R. Sethil, The complexity of flow shop and job-shop scheduling. Mathematics of Operations Research **1**. (1996), 117-129.

[2] D. L. Luo, S. X. Wu, M.Q. Li, and Z. Yang, Ant colony optimization with local search applied to the flexible job shop scheduling problems. ICCCAS conference in Communications, Circuits and Systems. (2008), 1015-1020.

[3] Wang Rui and Sun Bin, Design of Flexible Scheduling System Based on an Improved Genetic Algorithm. IC Computational Intelligence and Design, ISCID '09. Second International Symposium. Vol. **2**, (2009), 312-315.

[4] Xu Yaoqun and Wei Wenjian, Modified Adaptive Genetic Algorithm for Flexible Job-shop Scheduling Problem. Future Information Technology and Management Engineering (FITME) International Conference. Vol. **2**, (2010), 52-55.

[5] R. K. Suresh and K. M. Mohanasundaram, Pareto archived simulated annealing for job shop scheduling with multiple objectives. Cybernetics and Intelligent Systems, IEEE Conference. Vol. **2**, (2004), 712-717.

[6] Deroussi L. and da Fonseca J.B., A hybrid Ant Colony System for machine assignment problem in flexible manufacturing systems. Computer & Industrial Engineering, CIE09. International Conference. (2009), 205-210.

[7] Girish, B.S. and Jawahar N., A particle swarm optimization algorithm for flexible job shop scheduling problem. Automation Science and Engineering, CASE 2009, IEEE International Conference. (2009), 298-303.
[8] Nai-ping Hu and Pei-li Wang, An Algorithm for Solving Flexible Job Shop Scheduling Problems Based on Multi-objective Particle Swarm Optimization. Information Science and Engineering (ISISE), 2010 International Symposium. (2010), 507-511.
[9] W. Xia and Z. Wu, An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. Computers and Industrial Engineering. Vol. **48**, (2005), 409-425.
[10] N. B. Ho and J. C. Tay, Solving multiple-objective flexible job shop problems by evolution and local search. IEEE Trans. on Systems, Man, and Cybernetics, Part C. Vol. **38**, No. 5, (2008), 674-685.
[11] G.M.J.B. Chaslot, M.H.M. Winands, J.W.H.M. Uiterwijk, and H.J. Van Den Herik, Progressive strategies for Monte-Carlo tree search. In Information Sciences 07 Proceedings of the 10th Joint Conference. (2007), 655-661.
[12] L. Kocsis and C. Szepesvari, Bandit based Monte-Carlo planning. In 15th European Conference on Machine Learning. (2006), 282-293.
[13] S. Gelly, Y. Wang, R. Munos, and O. Teytaud, Modification of UCT with Patterns in Monte-Carlo Go. University of South Paris and Paris institute of technology. (2006).
[14] R. Agrawal, Sample mean based index policies with regret for the multi-armed bandit problem. Advances in Applied Probability. (1995), 1045-1078.
[15] H. Chen, J. Ihlow, and C. Lehmann, A genetic algorithm for flexible job-shop scheduling. IEEE International Conference on Robotics and Automation. Vol. **2**, (1999), 1120-1125.
[16] C. A. Coello and M. S. Lechuga, MOPSO: A proposal for multiple objective particle swarm optimization. Proc. Congress Evolutionary Computation. Vol. **1**, (2002), 1051-1056.
[17] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, A fast and elitist multi-objective genetic algorithm: NSGA-II. IEEE Trans. on Evolutionary Computation. Vol. **6**, No. 2, (2002), 182-197.

Chun-Liang Leu is an associate professor in Department of Information Technology at Ching Kuo Institute of Management and Health, Taiwan, Republic of China. He obtained the Ph.D. in Electrical Engineering from the Tamkang University, Taiwan, Republic of China. His current research focus is the optimization algorithm, data mining and artificial intelligence.

Shih-Yuan Chiu is a Ph.D. candidate in the Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien, Taiwan, and he is also an instructor in the Committee for General Education in National Dong Hwa University . His research interests include puzzle game solver and computational intelligence. He is also s 4-dan Go player.

Jiinpo Wu is an associate professor in the Department of Information Management, Tamkang University, Taiwan. He holds a PhD degree of Business Computer Information Systems from University of North Texas. His research interests currently focus on usage behavior of mobile users, NeuroIS, and virtual team management.

Li-Pen Chao received the MS degree in Information Management from Tamkang University. He is currently a PhD student of the Graduate Institute of Management Sciences, Tamkang University. His research interests are in the areas of brain and computer interface and activity theory.