Proceedings of the 2018 IEEE
International Conference on Robotics and Biomimetics
December 12-15, 2018, Kuala Lumpur, Malaysia

Dynamic Task Sequencing of Manipulator by Monte Carlo Tree Search

Yan-Wei Huang[1], Chong-Feng Liu[1], Su-Feng Hu[2], Zhang-Hua Fu[1], Yong-Quan Chen*[1]

***Abstract*— Nowadays manipulators are widely used in logistics systems, thus being important to improve the efficiency. In this paper, we transform the task sequencing problem of manipulator into a dynamic traveling salesman problem (DTSP), and use the Monte Carlo tree search (MCTS) approach to determine the execution order of the tasks. MCTS is an effective self-learning algorithm, which consists of four phases, i.e., initialization, simulation, selection and back propagation. Furthermore, to fit the dynamic feature of real-life applications, we introduce a dynamic mechanism using the information of historically found solutions. Finally, we carry out experiments based on 30 randomly generated task sets. The results show that, compared to the most popular method which executes all the tasks in sequence, MCTS is able to save 14.37%, 15.68%, 18.10% execution time (respectively on the small-scale, mid-scale and large-scale task sets), indicating its ability in improving the efficiency of manipulator.**

I. INTRODUCTION

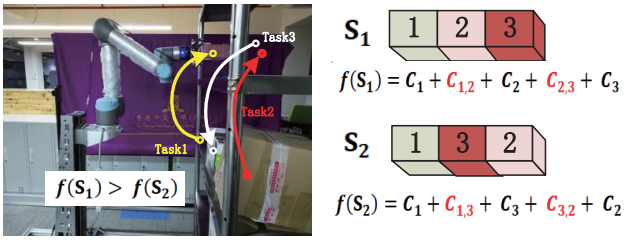
Manipulators are widely used in modern logistics systems. For example, in an intelligent warehouse, the sorting process is typically divided into the following steps: (1) the dispatch center receives tasks (customer orders) and assigns tasks to AGVs (Automated Guided Vehicle) and manipulators. (2) the AGVs move to the destination position and the manipulators pick products from shelves to AGVs. (3) AGVs move to the sorting stations, to finish the sorting task. During this process, a manipulator may continuously receive random task requests. Assuming that the tasks can be executed in arbitrary order, different execution order of tasks generally corresponds to different execution time. In this case, it is challenging to determine the execution order of tasks received by a manipulator, so as to minimize the total execution time.

The above problem is known as the task sequencing problem, which could be formally defined as follows: assume that a manipulator receives a series of random tasks respectively indexed by $1,2,\cdots,k,\cdots,n$ (received at time $t_1,t_2,\cdots,t_k,\cdots,t_n$ respectively), where task k requires the manipulator to move from begin position B_k to destination position D_k. The problem aims to determine the best execution order of all the tasks, i.e., determine a solution $S=(I_1,I_2,\cdots,I_k,\cdots,I_n)$, where I_k indicates the index of the task executed in the kth order. Furthermore, given the execution order, the start time T_{I_k} of each task I_k is determined as follows:

$$\begin{aligned} &T_{I_1} = t_{I_1}, \\ &T_{I_{k+1}} = Max(T_{I_k}+C_{I_k}+C_{I_k,I_{k+1}},\ \ t_{I_{k+1}}), \forall\ 1\le k\le n-1 \end{aligned} \tag{1}$$

Where C_{I_k} denotes the time cost of task I_k, i.e., the time needed by the manipulator to move from begin position B_{I_k} to destination position D_{I_k}. $C_{I_k,I_{k+1}}$ denotes the transition time cost between task I_k and I_{k+1}, i.e., the time needed by the manipulator to move from D_{I_k} to $B_{I_{k+1}}$. The objective is to minimize the total time cost (the finish time of the last task minus the receiving time of the first task), i.e.:

$$Minimize\ f(S) = T_{I_n}+C_{I_n}-t_1 \tag{2}$$



(a) (b)

Fig. 1. An example task sequencing problem of manipulator, where (a): information of the three tasks (b): two different solutions and the corresponding time cost.

In the example shown in Fig. 1, the manipulator receives three tasks respectively indexed by 1,2,3, where $S_1=(1,2,3)$ and $S_2=(1,3,2)$ are two solutions corresponding to different execution order and different time cost ($f(S_1)>f(S_2)$).

The task sequencing problem has many robotic-related applications [1]. Specifically, in the field of manipulators, the pose of a manipulator may affect its performance, thus making the problem much more complicated. In real-life applications, the most popular choice is to execute all tasks in sequence, according to the receiving time of each task. In 1989, Dubowsky and Blubaugh firstly defined the task sequencing problem of manipulator, based on various forms of distance matrices subject to the sequential constraints of tasks [2]. In 2014, Pardskevilh et al. took into account the diverse poses of manipulator [3], based on which Khelifa

*This work is partially supported by the National key R & D program of China (Grant No. 2017YFB1303701), partially supported by Economic, Trade and information Commission of Shenzhen Municipality (Grant No.: ZLZBCXLJZI20160805020016), and partially supported by the National Natural Science Foundation of China (Grant No: U1613216)

[1]The authors are with the Institute of Robotics and Intelligent Manufacturing, The Chinese University of Hong Kong, Shenzhen, China. y.w.h.robot@gmail.com(Y.W.Huang), liuchongfeng@cuhk.edu.cn (C.F. Liu), fuzhanghua@cuhk.edu.cn (Z.H. Fu), yqchen@cuhk.edu.cn (Y.Q. Chen, corresponding author)

[2]Su-Feng Hu is with the China Special Equipment Inspection and Research Institute allanhsf@hotmail.com

978-1-7281-0377-8/18/$31.00 © 2018 IEEE

Baizid further studied the influence of a manipulator's configuration on the execution efficiency [4]. From the point of operational research, this problem is equivalent to the famous traveling salesman problem (TSP). The classic TSP is described as follows: given a series of cities as well as the distance between any two cities, then starting from a given city, the problem aims to determine a route visiting every city once and finally returning to the starting city, so as to minimize the total distance. Specifically, in real-life applications of manipulators, the tasks (equivalent to cities) typically arrive randomly and the transition time cost between tasks (equivalent to the distance between cities) are generally unknown in advance, leading to a more complicated problem, i.e., the dynamic TSP (DTSP) [5]. For this problem, although numerous efforts were delivered during the past decades [6-9], there is still no algorithm which guarantees optimality within polynomial time. A simplified approach is to transform the DTSP into the classic TSP at first, and then use relevant methods (e.g., 2-Opt [10], 3-Opt [11]) to solve it. Another idea is to iteratively utilize the information discovered by local optimization to improve the efficiency of global search [12-14].

In this paper, we transform the task sequencing problem of manipulator into a DTSP problem, and originally develop a Monte Carlo tree search (MCTS) approach to solve it. MCTS is a meta-heuristic search algorithm based on the idea of reinforcement learning, which attempts to find high-quality solutions by iteratively generating solutions (simulation phase), selecting most promising solution (selection phase) and updating parameters (back propagation phase). By learning information from the solutions obtained during search, it is able to guide search towards promising region, thus improving the efficiency of search. Furthermore, to deal with the uncertainty of the DTSP, we introduce some dynamic search mechanism which utilizes the history information to guide the search. Finally, to verify the performance of MCTS, we carried out a lot of experiments using a real manipulator (UR5 brand), based on a series of randomly generated task sets. The results show that, on the small-scale (50 tasks), mid-scale (100 tasks) and large-scale (200 tasks) task sets, on average MCTS respectively saves 14.37%, 15.68%, 18.10% execution time, with respect to the most popular method which executes all the tasks in sequential order.

II. PROPOSED METHOD

A. Algorithm framework

In this paper we transform the task sequencing problem of manipulator into a DTSP problem, and develop a dynamic approach which consists of iterative rounds of MCTS to solve the problem. As shown in Fig. 2, the whole algorithm mainly includes the following steps: (1) Initialize the configuration of the manipulator, and perceive the surrounding environment; (2) if a new task is received, add it to the end of the task list; (3) in case the manipulator is idle and the task list is not empty, send the first task to the manipulator and delete it from the task list. (4) call MCTS (detailed in Fig. 3) to re-sort the execution order of the tasks in the task list (a round of MCTS terminates immediately in case the task list is changed). Repeat steps 2-4 until all the tasks are finished.

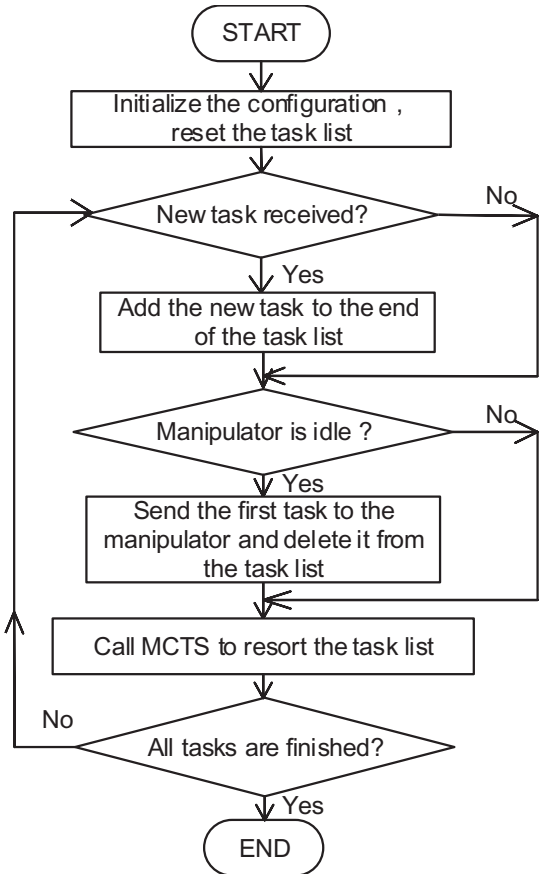


Fig. 2. The flow chart of the overall procedures

B. Estimation of C_{I_k} and $C_{I_k,I_{k+1}}$

Given the begin position B_{I_k} and destination position D_{I_k} of task I_k, we use the following method to estimate the time cost C_{I_k} needed to execute task I_k: (1) Firstly, we use the Open Motion Planning Library (OMPL) to generate a path (a series of discrete points) from B_{I_k} to D_{I_k} which satisfies the obstacle-avoid constraints. (2) Then, we use the Time Optimal Path Parameterization Algorithm (TOPP) [15] to estimate the interval time cost from each discrete point to the next one. (3) Finally, the time cost C_{I_k} is estimated by accumulating all the interval time cost.

Similarly, given the destination position D_{I_k} of task I_k and the begin position $B_{I_{k+1}}$ of the next task I_{k+1}, the transition time $C_{I_k,I_{k+1}}$ between task I_k and task I_{k+1}, i.e., the time needed by the manipulator to move from D_{I_k} to $B_{I_{k+1}}$, can be estimated in the same way. Given a number of tasks, all the values of C_{I_k} and $C_{I_k,I_{k+1}}$ are estimated and saved in a matrix C. Whenever the task list is changed (a task is added into or removed from the task list), the matrix C is updated dynamically, instead of re-calculating all the elements from scratch. With these information, in case the value of C_{I_k} or

$C_{I_k,I_{k+1}}$ is needed, we can fetch it directly from matrix C, to improve the efficiency of the search algorithm.

C. Implementation of MCTS

Given a task list containing a number of tasks, the MCTS method is called to schedule the execution order of the tasks, in order to save execution time. As shown in Fig. 3, the MCTS algorithm consists of four steps: (1) Initialization (define and initialize parameters); (2) Simulation (probabilistically generate solutions); (3) Selection (choose the best solution); (4) Back-propagation (update the parameters).

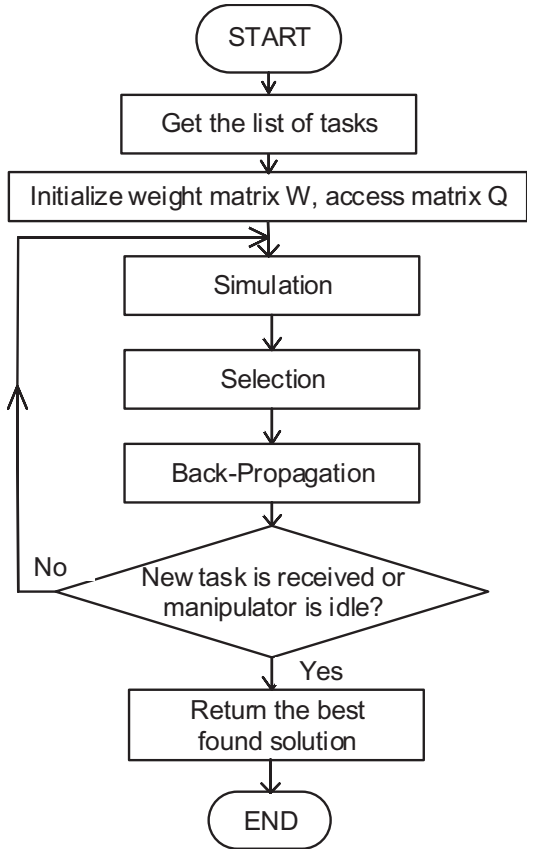


Fig. 3. The flow chart of MCTS

1) Initialization: Let $|TL|$ denote the length of the current task list, we define two $|TL| \times |TL|$ matrices, i. e., a weight matrix W whose element W_{ij} (all initialized to 1) is used to control the probability of selecting task j after task i during simulation, and an access matrix Q whose element Q_{ij} (all initialized to 0) is used to record the total times that task j is ordered after task i during simulation.

2) Simulation: Given a number of tasks, the simulation process attempts to probabilistically generate a number of different solutions. To generate each solution, it starts from an empty solution, and iteratively select a candidate task (whose execution order has not been fixed) as the next task to execute, until the execution order of every task is fixed. More clearly, assuming the index of the last selected task is i, the following formula is used to evaluate the potential of each candidate task j:

$$E_j = \frac{W_{ij}}{\Omega} + \alpha \sqrt{\frac{ln\ X}{Q_{ij}+1}} \tag{3}$$

Where Ω denotes the averaged W_{ij} value of all the candidate tasks, and X denotes the number of solutions already generated by simulation. In this formula, the left part $\frac{W_{ij}}{\Omega}$ aims to guide the search towards high-quality solutions found before (to enhance the intensification feature), while the right part $\sqrt{\frac{ln\ X}{Q_{ij}+1}}$ aims to guide the search towards unexplored region (to enhance the diversification feature). The term "+1" is used to avoid the possibility of a 0 denominator, and parameter α is used to achieve a balance between the intensification feature and diversification feature.

If more than one candidate tasks are available at a simulation step, we probabilistically select a candidate task as the next task. The probability P_j for selecting each candidate task j is determined as follows:

$$P_j = \frac{E_j}{\sum_{Candidate\ task\ k} E_k} \tag{4}$$

3) Selection: During the simulation process, we generate a number of (controlled by a parameter Z) random solutions. For each solution S, we estimate its objective value $f(S)$ based on the cost matrix C described in Section II-B. After that, we choose the solution with the lowest objective value as the best solution found during a round of simulation (denoted by $S_{best} = (I_1, I_2, \cdots, I_k, \cdots, I_n)$). Furthermore, based on the information of S_{best}, we call a back-propagation procedure to update the elements of matrices W and Q as follows.

4) Back-propagation: Corresponding to each pair of tasks I_k and I_{k+1} $(1 \le k \le n-1)$ belonging to S_{best} found above, we update the corresponding elements $W_{I_k,I_{k+1}}$ and $Q_{I_k,I_{k+1}}$ as follows:

$$W_{I_k,I_{k+1}} \leftarrow W_{I_k,I_{k+1}} + \beta + \gamma \times e^{\frac{f(S_{gbest}) - f(S_{best})}{f(S_{gbest})}} \tag{5}$$

$$Q_{I_k,I_{k+1}} \leftarrow Q_{I_k,I_{k+1}} + 1 \tag{6}$$

Where S_{gbest} denotes the global best solution found by MCTS (during iterative rounds of simulation) . β and γ are parameters used to control the increasing rate of $W_{I_k,I_{k+1}}$, respectively corresponding to linear part and nonlinear part.

As shown in Fig. 4, while optimizing by a round of MCTS, the simulation, selection and back-propagation steps are iterated, until the terminal condition (a new task is received or the manipulator is idle) is met. After that, it returns the best found solution, updates the task list, and turns into the next round of MCTS, to form a dynamic MCTS algorithm, as detailed in the following section.

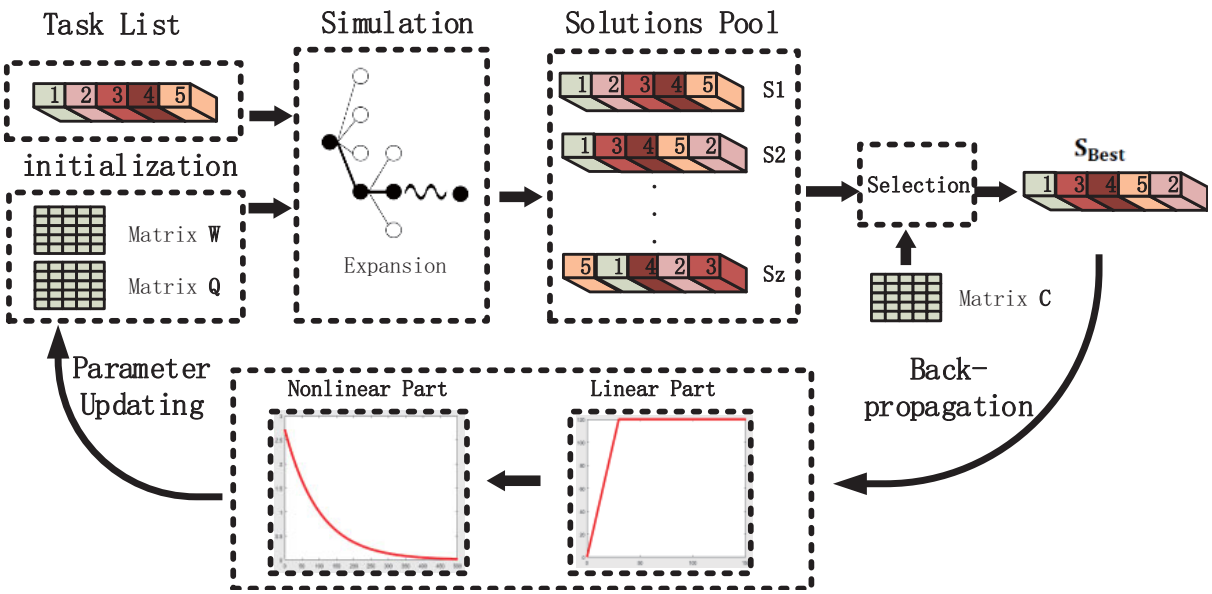


Fig. 4. The searching process of a round of MCTS.

III. Dynamic Monte Carlo Tree Search

The MCTS described above is suitable to tackle the problem in case the task list is fixed. However, in many real-life applications, new tasks are randomly (un-predictable) assigned to the manipulator and the task list changes dynamically. In this case, it is necessary to develop a dynamic variant of the MCTS. Due to this reason, whenever the task list is changed (a task is added into or deleted from the task list), we terminate the current round of MCTS, update the task list and matrix C, re-initialize the parameters, and then re-call a new round of MCTS to determine the execution order of all the tasks. This process is repeated, until all the tasks are finished (see Fig. 5 for an example of applying dynamic MCTS).

Specifically, in order to utilize the historically-discovered knowledge to improve the search efficiency, before rerunning MCTS we initialize the elements of weight matrix W as follows: if tasks I_k and I_{k+1} are adjacent in the best solution S_{gbest} found by the last round of MCTS, let $W_{I_k,I_{k+1}} \leftarrow W_{Init}$; otherwise, let $W_{I_k,I_{k+1}} \leftarrow 1$. Herein $W_{Init} > 1$ is a parameter used to control the influence of S_{gbest} to the performance of MCTS. Besides matrix W, all the remaining search mechanisms and parameters keep in accordance with the ones described in above subsections.

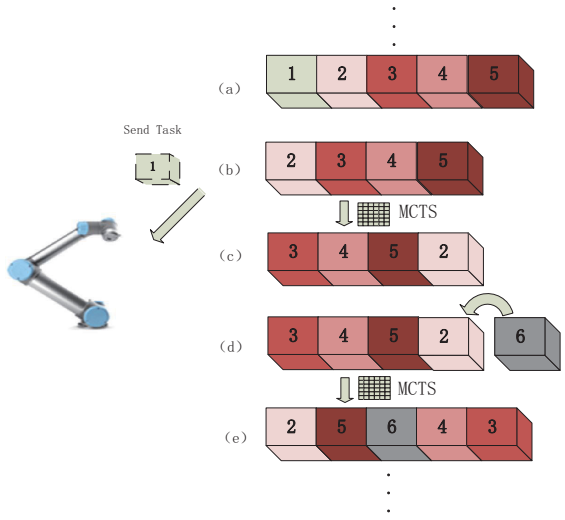


Fig. 5. An example of applying dynamic MCTS, where (a): the initial task list; (b): send the first task (task 1) to the manipulator and delete it from the task list; (c): re-call MCTS to re-sort the execution order of each task; (d): receive a new task (task 6) and add it to the end of the task list; (e): re-call MCTS to re-sort the execution order of each task.

IV. Experimental Results

A. Experimental protocol

In order to evaluate the performance of MCTS, we carry out experiments based on a real manipulator (the UR5 manipulator shown in Fig. 1). At first, we generate a series of task sets, each containing a number of random tasks (received dynamically). For each task, the start position and destination position are randomly given within the reachable space of the manipulator (on average about 15 seconds is needed by the manipulator to move from the start position to the destination position). Different task set corresponds to different number of tasks (classified into small-scale task set with 50 tasks, mid-scale task set with 100 tasks, large-scale task set with 200 tasks respectively) or different time interval between sequential tasks. More precisely, the time interval between any two sequential tasks is a random number which satisfies the normal distribution $N(\mu, \sigma^2)$, where μ denotes the average value of time interval (respectively set to 5 seconds, 10 seconds, 15 seconds, 20 seconds, 25 seconds), and σ^2 denotes the variance (respectively set to $\frac{\mu}{2}$ and $\frac{\mu}{5}$), leading to $3 \times 5 \times 2 = 30$ task sets.

TABLE I
PARAMETER SETTINGS

Parameter	Description	Value
Z	The number of solutions generated during simulation	3 $\|TL\|$
α	Used to guide the search direction, see Eq. (3)	2
β	The coefficient of linear part in Eq. (5)	$0.1\|TL\|$
γ	The coefficient of nonlinear part in Eq. (5)	1
W_{Init}	Used to control the influence of S_{gbest} (section III)	$0.5\|TL\|$

In industrial applications, the most popular method is to execute all the tasks in sequence (denote this method by SEQ), ordered by the receiving time of each task. In order to demonstrate the superiority of MCTS with respect to SEQ, for each task set, we execute the tasks on a UR5 manipulator following the order respectively determined by SEQ and MCTS. After that, we record and compare the total execution time of SEQ and MCTS, respectively on the small-scale, mid-scale and large-scale task sets, as detailed in Table 2 to Table 4. Note that both SEQ and MCTS algorithms are coded in C++ language, using the Robot Operating System (ROS), and run on a computer with an Intel Corei7-7700 3.6 GHz CPU and 15.6 GB memory.

B. Parameters setting

The proposed MCTS relies on a series of parameters, which may affect the performance of the proposed algorithm. After careful tuning, we choose the values listed in Table 1 as default settings of the parameters, where $|TL|$ denotes the length of the current task list.

C. Results and analysis

For each of the 30 task sets generated above, we respectively use SEQ and MCTS to determine the execution order of the tasks, and then execute them on a UR5 manipulator. After that, we summarize the results in Tables 2-4, respectively corresponding to small-scale, mid-scale and large-scale task sets. In each table, the first column indicate the number of tasks in each task set. Columns 2-3 respectively denote the average value (in seconds) and variance of the normal distribution (related with the time interval between tasks). Columns 4 gives the running time (in seconds) elapsed by SEQ to execute all the tasks of each task set, while column 5 gives the same information corresponding to MCTS. Column 6 gives the save percentage in terms of running time (MCTS vs. SEQ). Finally, the last row gives the average results over the task sets, while meaningless items are marked as '-'.

From the tables, we observe that on the small-scale, mid-scale and large-scale task sets, MCTS respectively saves 14.37%, 15.68%, 18.10% execution time compared to SEQ. These results clearly demonstrate the ability of MCTS in improving the efficiency of manipulator. Furthermore, we observe from the results that on the large-scale task sets, the improve rate is generally larger than the improve rate on the small-scale task sets, indicating the importance of MCTS on the large-scale instances.

TABLE II
EXPERIMENTAL RESULTS ON 10 SMALL-SCALE TASK SETS

n	$\mu(s)$	σ^2	T_{SEQ} (s)	T_{MCTS} (s)	$Save(\%)$
50	5	$\mu/2$	1617.05	1461.26	9.63
50	5	$\mu/5$	1697.87	1392.27	17.99
50	10	$\mu/2$	1774.25	1431.35	19.32
50	10	$\mu/5$	1656.62	1404.83	15.19
50	15	$\mu/2$	1635.45	1408.76	13.86
50	15	$\mu/5$	1651.12	1446.28	12.40
50	20	$\mu/2$	1562.48	1440.77	9.96
50	20	$\mu/5$	1575.39	1404.57	21.57
50	25	$\mu/2$	1691.48	1534.58	9.27
50	25	$\mu/5$	1662.65	1420.98	14.53
Average	-	-	1677.78	1434.565	14.37

TABLE III
EXPERIMENTAL RESULTS ON 10 MID-SCALE TASK SETS

n	$\mu(s)$	σ^2	T_{SEQ} (s)	T_{MCTS} (s)	$Save(\%)$
100	5	$\mu/2$	3414.65	2676.76	21.60
100	5	$\mu/5$	3354.76	2643.13	21.21
100	10	$\mu/2$	3328.03	2744.76	17.52
100	10	$\mu/5$	3303.8	2763.9	16.34
100	15	$\mu/2$	3363.61	2767.96	17.70
100	15	$\mu/5$	3365.84	2877.13	14.51
100	20	$\mu/2$	3206.07	2781.15	13.25
100	20	$\mu/5$	3107.58	2739.89	11.83
100	25	$\mu/2$	3254.81	2878.79	11.55
100	25	$\mu/5$	3315.39	2962.53	10.64
Average	-	-	3301.45	2783.6	15.68

TABLE IV
EXPERIMENTAL RESULTS ON 10 LARGE-SCALE TASK SETS

n	$\mu(s)$	σ^2	T_{SEQ} (s)	T_{MCTS} (s)	$Save(\%)$
200	5	$\mu/2$	6441.14	5017.15	22.10
200	5	$\mu/5$	6118.31	4958.45	18.95
200	10	$\mu/2$	6251.69	5024.93	19.62
200	10	$\mu/5$	6222.13	5040.91	18.98
200	15	$\mu/2$	6445.84	5198.89	19.34
200	15	$\mu/5$	6239.57	4985.75	20.09
200	20	$\mu/2$	6321.28	5345.66	15.43
200	20	$\mu/5$	6777.77	5624.85	17.01
200	25	$\mu/2$	6791.25	5846.8	13.90
200	25	$\mu/5$	6552.19	5533.04	15.55
Average	-	-	6416.12	5257.643	18.10

V. CONCLUSION

In this paper we show that by using Monte Carlo tree search (MCTS) approach to determine the execution order of dynamic tasks, it is able to remarkably improve the efficiency of manipulator, compared to the traditional method which executes all the tasks in sequence. In the future, we will try to extend the work to industrial applications with manipulators, to improve the efficiency of real-life systems.

REFERENCES

[1] Spensieri D, Carlson JS, Bohlin R, Soderberg R. Integrating assembly design, sequence optimization, and advanced path planning. InASME 2008 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference 2008 Jan 1 (pp.73-81). American Society of Mechanical Engineers.in Plastics, 2nd ed. vol. 3, J. Peters, Ed. New York: McGraw-Hill, 1964, pp. 15-64.

[2] Dubowsky S, Blubaugh TD. Planning time-optimal robotic manipulator motions and work places for point-to-point tasks. IEEE Transactions on Robotics and Automation. 1989 Jun;5(3):377-81.

[3] Zacharia PT, Aspragathos NA. Optimization of industrial manipulators cycle time based on genetic algorithms. InIndustrial Informatics, 2004. INDIN'04. 2004 2nd IEEE International Conference on 2004 Jun 26 (pp. 517-522).

[4] Baizid K, Meddahi A, Yousnadj A, Chellali R, Khan H, Iqbal J.Robotized task time scheduling and optimization based on Genetic Algorithms for non redundant industrial manipulators. In Robotic and

Sensors Environments (ROSE), 2014 IEEE International Symposium on 2014 Oct 16 (pp. 112-117).
[5] Guntsch M, Middendorf M. Pheromone modification strategies for ant algorithms applied to dynamic TSP. In Workshops on Applications of Evolutionary Computation 2001 Apr 18 (pp. 213-222). Springer, Berlin, Heidelberg.
[6] Angus D, Hendtlass T. Ant colony optimisation applied to a dynamically changing problem. In International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems 2002 Jun 17 (pp. 618-627). Springer, Berlin, Heidelberg
[7] Guntsch M, Middendorf M. Applying population based ACO to dynamic optimization problems. In International Workshop on Ant Algorithms 2002 Sep 12 (pp. 111-122). Springer, Berlin, Heidelberg.
[8] Kang L, Zhou A, McKay RI, Li Y, Kang Z. Benchmarking algorithms for dynamic travelling salesman problems. In IEEE Congress on Evolutionary Computation 2004 Jun 19 (pp. 1286-1292)
[9] Li C, Yang M, Kang L. A new approach to solving dynamic traveling salesman problems. In Asia-Pacific Conference on Simulated Evolution and Learning 2006 Oct 15 (pp. 236-243). Springer, Berlin, Heidelberg
[10] Croes GA. A method for solving traveling-salesman problems. Operations research. 1958 Dec;6(6):791-812
[11] Lin S. Computer solutions of the traveling salesman problem. Bell System Technical Journal. 1965 Dec;44(10):2245-69
[12] Cruz C, Gonzlez JR, Pelta DA. Optimization in dynamic environments: a survey on problems, methods and measures. Soft Computing. 2011 Jul 1;15(7):1427-48
[13] Yang S, Jiang Y, Nguyen TT. Metaheuristics for dynamic combinatorial optimization problems. IMA Journal of Management Mathematics. 2012 Oct 10;24(4):451-80
[14] Branke J. Memory enhanced evolutionary algorithms for changing optimization problems. In Evolutionary Computation, 1999. CEC 99.Proceedings of the 1999 Congress on 1999 (Vol. 3, pp. 1875-1882).
[15] Pham QC. A general, fast, and robust implementation of the time-optimal path parameterization algorithm. IEEE Transactions on Robotics. 2014 Dec;30(6):1533-40.