Monte Carlo Tree Search with Branch and Bound for Multi-Robot Task Allocation

Bilal Kartal
NVIDIA
23 PUBLICATIONS **324** CITATIONS
SEE PROFILE

Ernesto queiros Nunes
University of Minnesota Twin Cities
14 PUBLICATIONS **257** CITATIONS
SEE PROFILE

Julio Godoy
University of Minnesota Twin Cities
21 PUBLICATIONS **165** CITATIONS
SEE PROFILE

Maria Gini
University of Minnesota Twin Cities
344 PUBLICATIONS **4,888** CITATIONS
SEE PROFILE

Monte Carlo Tree Search with Branch and Bound for Multi-Robot Task Allocation

Bilal Kartal, Ernesto Nunes, Julio Godoy, and Maria Gini
Department of Computer Science and Engineering
University of Minnesota
(bilal,enunes,godoy,gini)@cs.umn.edu

Abstract

Multi-robot teams are effective in a variety of task allocation domains such as warehouse automation and surveillance. Robots in such domains have to perform tasks at given locations and specific times. Tasks have to be allocated to optimize given team objectives, such as minimizing the total distance traveled. We propose an efficient, satisficing and centralized Monte Carlo Tree Search (MCTS) based algorithm which exploits the branch and bound paradigm with a novel search parallelization method to solve the multi-robot task allocation problem with spatial, temporal and other side constraints. Unlike previous heuristics proposed for this problem, our approach maintains asymptotic convergence guarantees of MCTS and it has efficient anytime behavior. It finds near-optimal solutions for non-trivial problems in the Solomon data sets in an hour.

1 Introduction

Autonomous service robots are employed for a variety of of tasks such as home care for elderly [Portugal *et al.*, 2015], escorting visitors on multi-floor buildings [Veloso *et al.*, 2012], and material transportation within hospitals [Özkil *et al.*, 2009]. These problems necessitate a high level task allocation planner for efficiency. In this work, we study the multi-robot task allocation (MRTA) problem with temporal and capacity constraints with homogeneous robots (henceforth MRTA-TW). This problem falls under the category single-task robot, single-robot task, time extended allocation with in-schedule dependencies ID[ST-SR-TA] [Korsah *et al.*, 2013], although the constraints herein considered are far more complex than the ones specified in the taxonomy.

In MRTA-TW, a fixed set of tasks is allocated to a team of robots such that the total distance traveled by all robots is minimized, and temporal and other side constraints are satisfied. This problem is $\mathcal{NP}$-hard, even when a single-robot is considered. The single-robot version of the problem is a variant of the elementary shortest path problem with resource constraints, which is a well-known $\mathcal{NP}$-hard problem. For this reason, optimal methods are not practical for datasets involving tens of robots and hundreds of tasks due to large run times (several hours to days for hundreds of tasks and tens of robots). Hence, efficient but suboptimal solutions are needed.

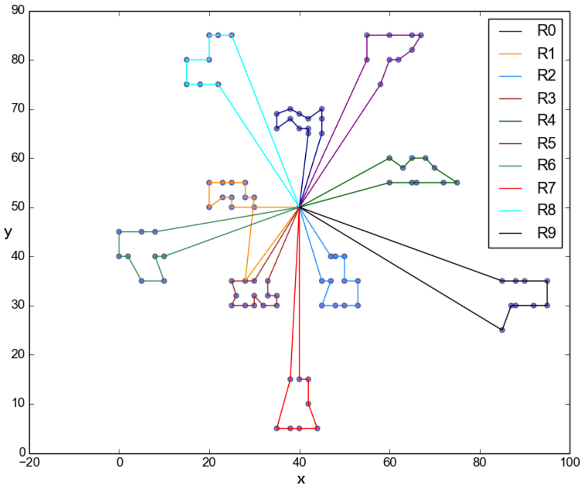


Figure 1: A near-optimal solution generated by MCTS for C101 test scenario in Solomon benchmark. This solution has an approximation rate of 1.03 to an optimal one. $R_i, i \in \{0, 1, 2, \ldots, 9\}$ indicate the robots.

Approximate centralized and decentralized methods have been proposed for MRTA-TW problems (e.g. [Gombolay *et al.*, 2013; Ponda *et al.*, 2010; Nunes and Gini, 2015; McIntire *et al.*, 2016]). While there exist some efficient schedulers (e.g. [Gombolay *et al.*, 2013]) for problems in which distances are small enough to be subsumed into tasks' durations, little attention has been given to solvers for problems in which routing and scheduling problems have to be solved simultaneously. Our approach tackles the latter problem and provides a solver that achieves asymptotic optimality, while attaining optimal or near-optimal results for non-trivial number of tasks and robots within practical runtimes.

Our main contribution is a centralized heuristic that balances between efficiency and solution quality by employing a modified Monte Carlo Tree Search (MCTS) approach. MCTS is an anytime sampling based technique employing Upper Confidence Bounds (UCB) [Auer *et al.*, 2002] to balance exploration vs. exploitation. MCTS with UCB has been proven

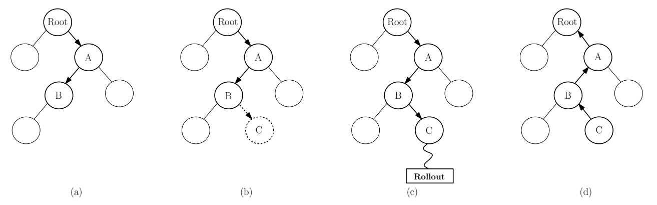


Figure 2: Overview of Monte Carlo Tree Search: (a) *Selection*: UCB equation is used recursively until a node with an unexplored action is selected. Assume that nodes A and B are selected. (b) *Expansion*: Node C is added to the tree. (c) *Random Rollout*: A sequence of random actions is taken from node C to complete the partial path. (d) *Back-propagation*: Full path is evaluated and the score is back-propagated from node C to the root.

to converge to an optimal solution for finite horizon problems [Kocsis and Szepesvári, 2006].

The main motivation for extending MCTS stems from the method's recent success in handling problems with large state spaces [Enzenberger *et al.*, 2010]. To apply MCTS to MRTA-TW, we improve upon MCTS in three fundamental ways: (1) We design a multi-objective evaluation function that balances between minimizing the distance traveled by all the robots and maximizing the task completion rate. (2) We propose a customized root parallelization scheme that helps the algorithm better explore the solution space. (3) We use branch and bound to prune parts of the search tree that do not improve upon the incumbent solution.

Our solution's quality is assessed empirically using the Solomon data set [Solomon, 1987] for vehicle routing problems with time-windows (VRPTW), and compared with many optimal results from VRPTW. Our method finds solutions that are at most 1.6 away from optimal ones, and achieve an average of 98% task completion rate across all data sets.

2 Background

Our work draws knowledge from the multi-robot task allocation (MRTA) with time windows and the MCTS literature.

2.1 Centralized Methods for MRTA-TW

There have been previous attempts to deal with variants of our problem in the MRTA literature. Alighanbari et al. [2003] studied the allocation of tasks that have to be done at a certain location and have constraints on their start time. They offer a Mixed Integer Linear Programming (MILP)-based solution and a more scalable tabu-search algorithm. Unlike our problem, they only consider precedence ordering of tasks, and not hard temporal constraints on tasks.

Koes and colleagues [2005] proposed a MILP-based heuristic for allocating tasks with deadline constraints to heterogeneous robots. They consider only problems where task types and robot capabilities have to match. Like in [Alighanbari *et al.*, 2003], their MILP-based approach does not scale to large number of tasks and robots.

More recently, Gombolay et al. [2013] studied allocating tasks with spatiotemporal constraints to robots, and offered a MILP-based efficient heuristic that performed well for up to 100 tasks. Their work considers location constraints that prevent robots from working too close together. Task allocation with resource contention has been recently studied in [Nam and Shell, 2015] as well. The closest work to ours is by Korsah et al. [2010], who explore the vehicle routing problem with location choice. The main difference between the latter work and ours is that they include precedence constraints. Precedence constraints impose structure on the ordering of tasks. With pre-processing, the ordering would reduce the size of our search space.

Our problem can also be cast as a VRPTW [Solomon, 1987]. There is a rich literature of centralized methods for VRPTW [Bräysy and Gendreau, 2005a; 2005b]. Most of the heuristic methods that yield good quality solutions efficiently require modeling and parameter tuning specific to a data set, which makes these methods hard to generalize. Instead, our approach, which is based on Monte Carlo Tree Search, is more general and is guaranteed, given enough time, to converge to an optimal solution.

2.2 Monte Carlo Tree Search

Monte Carlo Tree Search, illustrated in Figure 2, is a best first search algorithm that has gained traction after its breakthrough performance in the game Go [Coulom, 2006]. Recently an MCTS based distributed approach combined with deep neural networks defeated the strongest Go player in the world on a full size Go board [Silver *et al.*, 2016]. Other than games [Lisỳ *et al.*, 2015], MCTS has been employed for a variety of domains such as demand management for smart grid systems [Galvan-Lopez *et al.*, 2014], single vehicle transportation planning [Edelkamp and Gath, 2014], multi-robot patrolling [Kartal *et al.*, 2015], multi-agent narrative generation [Kartal *et al.*, 2014], and Sokoban puzzle generation [Kartal *et al.*, 2016]. Our use of MCTS follows the approach in [Kocsis and Szepesvári, 2006], which employs the UCB (Upper Confidence Bounds) technique to balance exploration vs. exploitation during planning.

3 MCTS for Multi-Robot Task Allocation

We formalize the MRTA-TW problem and describe how we adapt MCTS to address it. We introduce our proposed evaluation function and present how branch and bound paradigm can be applied within the MCTS algorithm, along with a simple but novel search parallelization method.

3.1 Problem Formulation

In the MRTA-TW problem, there are m tasks which need to be allocated to n robots. The objective is to minimize the total distance that the robots must travel to reach the m tasks while satisfying the temporal and capacity constraints. We assume $n \leq m$. Each task has a x-y location, service time, capacity demand, and time window. Demand is the number of capacity units consumed. A time window specifies the temporal constraints for task execution, by specifying the earliest time to start the task, and the latest time to end it. Time windows can have different lengths and are allowed to overlap.

We assume point mass robots, each with an x-y coordinate, a capacity, and a global deadline by which it has to return to a

Algorithm 1: MCTS with Search Parallelization

```
Input  : Budget
Output: Policy
$D(\hat{\pi_{best}}) = \infty$ ;
ThreadId = get_thread_num();
bestLocalScore[ThreadId] = $\infty$ ;
while timeElapsed $\leq Budget$ do
    Expanded_Node $\leftarrow$ ucbSelection(root[ThreadId]) ;
    $\hat{\pi} \leftarrow$ rollout(Expanded_Node) ;
    if $f(\hat{\pi}) >$ bestLocalScore[ThreadId] then
        bestLocalScore[ThreadId] = $f(\hat{\pi})$;
        bestLocalPolicy[ThreadId] = $\hat{\pi}$;
        if $m' = m$ AND $D(\hat{\pi}) < D(\hat{\pi_{best}})$ then
            set_lock(&writelock);
            if $D(\hat{\pi}) < D(\hat{\pi_{best}})$ then
                $D(\hat{\pi_{best}}) = D(\hat{\pi})$;
                $\hat{\pi_{best}} = \hat{\pi}$;
            end
            unset_lock(&writelock);
        end
    end
    backpropagate($f(\hat{\pi})$) ;
end
```

designated depot. We assume that a task needs only one robot to be executed, and a robot can perform several tasks, one at a time. Hence, the ultimate goal is to compute a route for each robot that starts and ends at the depot. A robot assigned to a route should be able to execute each task within the task's time window, and have enough time to travel between tasks and return to the depot. Each task should be included in only one route, and routes should cover all the tasks.

The problem is modeled as a directed graph, $G = (V, E)$, where the vertices V are the locations of the tasks and the depot. The set of weighted edges E includes only feasible edges obtained from pairing vertices in V. Edge $e_a^b \in E$ is a feasible edge if and only if task b can be completed after task a without violating the time-window constraints. The weight on the edges represent the distance between the pair of tasks. In our MCTS model routes are turned into allocation policies.

Let $\pi_i = \{\{r_i, t_i^1\}, \{r_i, t_i^2\}, ..., \{r_i, t_z^{|\pi_i|}\}\}$ denote the *individual task allocation policy* of robot r_i, where t_i^j corresponds to the j-th allocated task in the policy of robot r_i and t_z corresponds to the depot. For each employed robot, the *return to depot* action is included as a dummy task. Let $\hat{\pi} = \{\pi_1 \cup \pi_2 \cup .... \cup \pi_n\}$ denote the *global task allocation policy* for the entire set of robots, where each task is allocated to a single robot. In addition, let $D(\hat{\pi})$ represent the total distance traveled by the robots while following the policy $\hat{\pi}$. A complete policy is one where all tasks are allocated, i.e., $|\hat{\pi}| = m + n'$, while in an incomplete policy some tasks remain unallocated, i.e., $|\hat{\pi}| < m+n'$ where $n' \leq n$ is the number of robots that performed at least one task. Our objective is to find a *complete* policy $\hat{\pi}$ such that $D(\hat{\pi})$ is *minimized.* Once the search is over, the solution $\hat{\pi}$ is decomposed into n individual robot policies to be executed simultaneously.

3.2 Approach Overview

We formulate the MRTA-TW problem as a tree search and propose an MCTS based method as outlined in Alg. 1. In our tree structure, robots are employed sequentially, i.e., the route for robot r_{i+1} will be computed once robot r_i returns to the depot.

At each level of the tree, a single robot is assigned one of the remaining tasks or returns to the depot. Once a robot returns to the depot it cannot be allocated any other task. This creates unbalanced allocations, however it keeps the branching factor at a manageable size.

During the search, the UCB algorithm is used to choose which task to allocate to each robot, as shown in Eqn. 1. Each parent node p chooses its child s with the largest $UCB(s)$ value. Here, $w(.)$ denotes the average evaluation score obtained by Eqn. 2, $\hat{\pi}_s$ is the parent's policy, which is updated to include child node s; p_v is visit count of parent node p, and s_v is visit count of child node s.

$$UCB(s) = w(\hat{\pi}_s) + C \times \sqrt{\frac{\ln p_v}{s_v}} \tag{1}$$

If a node with at least one unexplored child is reached ($s_v = 0$), a new node is created for one of the unexplored action sets. After the rollout and back-propagation steps, the selection step is restarted from the root again. This way, the tree can grow in an uneven manner, biased towards better solutions. The value of C determines the rate of exploration, where smaller C implies less exploration. $C = \sqrt{2}$ is necessary for search completeness of MCTS.

Given that our problem formulation is completely deterministic, we can store the best found solution during the search and optionally halt the search with some solution quality threshold by exploiting the anytime property of MCTS.

3.3 Policy Evaluation Function

The MCTS algorithm needs an evaluation function to estimate the true rewards of tree actions by measuring the quality of full policies extended with random rollout actions. Evaluation functions are straightforward in most games where the player gets a payoff of 1, 0.5, and 0 for winning, tying, and losing, respectively. Evaluation is more complex for MRTA-TW because neither the value of the optimal solution is known nor most candidate solutions allocate all the tasks. The evaluation function in (2) helps find complete allocations that take distance and allocation percentage into account.

Let m' represent the number of allocated tasks for task allocation policy $\hat{\pi}$. Due to the nature of highly constrained properties of tasks, most candidate solutions fail to allocate all the tasks. However, MCTS still needs to direct the search to find better solutions. In this work, we propose an anytime policy evaluation function which guarantees that given more planning time, MCTS will find better solutions. Let α denote a loose upper bound on the total traveled distance $D(\hat{\pi})$, and $\hat{E}$ be a sorted version of E (in descending order of their distances). Then, $\alpha = 2 \times \sum_{i=1}^{m+n} e_i$ where $e_i \in \hat{E}$. Let δ denote task completion of the candidate policy where $\delta = 1$ if $m' = m$, and $\delta = 0.5$ otherwise.

To discourage incomplete policies, we define a negative reward parameter, ψ, which is computed as follows: $\psi = 2 \times \sum_{i=1}^{m-m'} e_i$, $e_i \in E'$, where $E' \subset E$ is the set of edges directed from completed to uncompleted tasks, from uncompleted to uncompleted tasks, and from uncompleted tasks to the depot sorted in descending order of distances. While calculating ψ, we use the simple observation that no edge in graph G will be traveled twice.

Based on these definitions, we propose an evaluation function $f(\hat{\pi})$, to assess policy $\hat{\pi}$, as follows:

$$f(\hat{\pi}) = \frac{\alpha - (D(\hat{\pi}) + \psi)}{\alpha} \times \delta \tag{2}$$

Our evaluation function considers the worst case insertion cost of the unfulfilled tasks from the remaining feasible edges for penalty term. This helps MCTS to avoid getting trapped by partial policies with small total distances which are unlikely to complete all tasks.

Properties of the Policy Evaluation Function

Our evaluation function guarantees that $f(\hat{\pi_a}) > f(\hat{\pi_b})$ holds if $\hat{\pi_a}$ completes all the tasks and $\hat{\pi_b}$ misses at least one task. To show this, we can simply show bounds of $f(\hat{\pi_a})$ and $f(\hat{\pi_b})$ separately, which would imply our claim.

Case 1: $0.5 \leq f(\hat{\pi_a}) < 1$. We can observe that $D(\hat{\pi}) \leq \alpha/2$ is true as α is accumulated from the longest $m + n$ edges on graph G and multiplied by 2, while $|\hat{\pi}| \leq m + n$ depending on the number of robots utilized. Therefore, $\psi = 0$ and $\delta = 1$ holds by completion of all the tasks which completes our claim for case 1.

Case 2: $0 < f(\hat{\pi_b}) < 0.5$. This holds true because by missing at least one task δ will be set to 0.5, and $D(\hat{\pi}) + \psi > 0$ holds for any policy. The δ parameter behaves like a step function.

Lastly, our evaluation function returns monotonically increasing values along any path from the tree root as more tasks are allocated. For every allocated task, the actual traveled distance increases by some $c \geq 0$, while ψ decreases by at least c; this guarantees that the evaluation score of the policy never decreases for any sequence of actions in the tree as more tasks are completed.

3.4 Application of Branch and Bound

Branch and bound is used to prune nodes during search, based on the incumbent solution that allocates all the tasks. A simple observation is that no matter how many robots are utilized, each edge in E will be traveled at most once. We let $D(\hat{\pi_{best}})$ denote the total distance traveled corresponding to the best solution found so far that completes all the tasks.

Given a partial task allocation policy π_p which completes m_p tasks using n_p robots with a total distance of $D(\hat{\pi_p})$, there are two cases to consider. First, let's assume that all remaining robots, $n - n_p$, are at the depot. Then we generate a possible edge list, $E_{rest} \subset E$ by considering all edges from the depot to the uncompleted tasks, from uncompleted to uncompleted tasks, and from uncompleted tasks to the depot. Second, if there is a robot that completed a task but did not return to depot yet, we also consider edges from the robot location to uncompleted tasks and to depot for E_{rest}. In case there is no robot at depot, we neglect edges directed from the depot. During bounding tree branches, if there are fewer robots than tasks, we assume that each robot has to complete at least one task while a lower-bound on the future distance to travel is computed along tree branches.

We branch new nodes if the following condition holds $D(\hat{\pi_{best}}) - D(\hat{\pi_p}) > \sum_{i=1}^{q} e_i$ where $e_i \in E_{rest}$ and E_{rest} is sorted in ascending order. Here, q is the minimum number of edges to cover to complete a partial allocation policy, which is computed as follows; if $n < m$, then $q = (m + n) - |\pi_p|$, otherwise $q = m - m_p$.

3.5 Parameterized Root Parallelization for MCTS

Several different search parallelization methods have been proposed for MCTS, such as tree parallelization, leaf parallelization and root (or single run) parallelization, as summarized in [Browne *et al.*, 2012]. Among these different approaches, root parallelization has been shown to perform best for the game Go [Chaslot *et al.*, 2008]. Fern at al. [2011] show the effectiveness of root parallelization by exhaustive experiments over different problems. Root parallelization simply creates multiple search trees, one per thread, and merges the search trees once the search budget is complete to generate policies. It has minimal overhead as the threads do not communicate until the merging step.

We propose a novel variant of root parallelization, henceforth *parameterized root parallelization*. Each tree is given a different UCB exploration parameter so that they can explore the search space in different ways. Similar to pure root parallelization, our approach also creates multiple independent search trees per thread. There is a globally shared variable, $D(\hat{\pi_{best}})$, keeping the best complete solution distance found by any tree. The design idea for our approach is that finding a locally optimal solution early can better calibrate the search direction for all threads given the very large state space. This also improves memory efficiency as we can prune existing nodes that are guaranteed to not beat the best found so far.

For k cores, we assign exploration parameters as follows; $\hat{C} = \{\frac{2C}{k}, \frac{4C}{k}, \ldots, C, 2C, 3C \ldots, (\frac{k}{2} + 1)C\}$ where $C = \sqrt{2}$. The first half of the cores are assigned smaller exploration parameters so that they can search deeper early to find a complete solution to expedite pruning in all trees, while the second half of the cores will explore more so as not to get trapped by a local maximum. In cases where none of the threads is able to find a task allocation policy which completes all the tasks, the result of our parallelization approach is identical to running MCTS k times with different C values with a single core and returning the best found solution.

4 Experimental Setup and Results

We assessed our MCTS method using the Solomon dataset for vehicle routing on commodity hardware. We used 8 logical cores on an Intel Core i7-4790 3.6 GHz Quad-Core computer with 32GB RAM. The algorithm is run once for each data instance with search times set to 1 hour. We present a near-optimal solution generated by our approach in Figure 1. We show how branch and bound improves our results for all categories in Figure 3. Overall anytime behavior of our approach is presented in Figure 4.

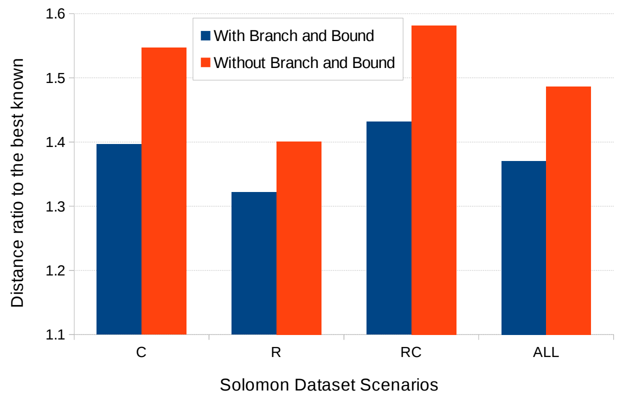


Figure 3: Smaller is better. Embedding branch and bound in the MCTS algorithm improves the overall distance ratio to the best known solutions 11% across all instances in the Solomon data set.

The Solomon dataset provides a rich variety of problem scenarios for task locations and time windows. Tasks are either clustered (C), randomly scattered (R), or a mix of clustered and randomly scattered (RC). Each of these categories has either tight time windows (type 1) or large time windows (type 2). The individual spatiotemporal categories (e.g. C1, R2) have between 8-12 individual instances.

Each instance has 100 tasks, each task has a uniformly generated demand drawn from $\mathcal{U}(1, 50)$, and service times of either 10 (C and RC data instances) or 90 (R data instances). The earliest start time for the time windows is drawn from $\mathcal{U}(1, 1000)$ and $\mathcal{U}(1, 3400)$ for types 1 and 2 data instances, respectively. The latest finishing times are drawn from $\mathcal{U}(20, 1000)$. The x-y coordinates of the tasks and robots are uniformly drawn from $\mathcal{U}(0, 100)$. The global deadlines for the robots have values between ~ 200 and ~ 3500.

4.1 Comparison to other methods

As no single method produces the best results for all problem instances in Solomon benchmark, we compare our results with the best known results [Solomon, 2005] obtained from up to 16 methods including metaheuristics, local search, ant-based, and genetic algorithms. While these methods are more efficient, unlike our method, they neither generalize well across data sets nor do they provide any guarantees. We summarize our results in Table 1 and Table 2.

4.2 Fixed number of robots

In this set of experiments the number robots used is upper bounded by the number of robots used in the best known solutions from the VRPTW literature.

Task allocation percentage results for type 1 and 2 data instances are presented in Figure 4. Our algorithm allocates more than 40% of the tasks after 5 minutes; it allocates nearly all tasks across all datasets within one hour. The average task allocation rate and the percentage of instances in which all tasks are allocated are reported in Table 2. The algorithm yields high average allocation rates across all datasets, and it attains relatively lower percentages of instances in which 100% of tasks are allocated.

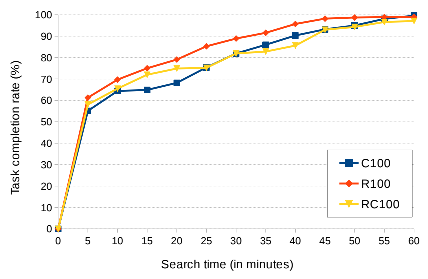
Scenario	MCTS	Optimal	Task %	Ratio to Opt.
C101	853.5	827.3	1	1.03
C102	1287.7	827.3	1	1.55
C103	1320.5	826.3	1	1.59
C104	1249.9	822.9	1	1.51
C105	1038.2	827.3	1	1.25
C106	982.6	827.3	0.99	-
C107	1117	827.3	0.98	-
C108	1076.1	827.3	0.99	-
C109	1173.9	827.3	1	1.42
R101	1820.8	1637.7	1	1.11
R102	1716.5	1466.6	1	1.17
R103	1593.4	1208.7	1	1.31
R104	1303.1	971.5	1	1.34
R105	1640.9	1355.3	1	1.21
R106	1532.9	1234.6	1	1.24
R107	1363.9	1064.6	0.99	-
R108	1051	960.9	0.96	-
R109	1428.6	1146.9	1	1.25
R110	1381.8	1068	1	1.29
R111	1436.5	1048.7	1	1.37
R112	1055.2	982.1	0.93	-
RC101	1515.2	1619.8	0.96	-
RC102	1851.4	1457.4	1	1.27
RC103	1554.2	1258	0.98	-
RC104	1373.7	1135.5	0.94	-
RC105	1973	1513.7	1	1.30
RC106	1520.8	1424.7	0.91	-
RC107	1614.1	1207.8	0.99	-
RC108	1515.2	1114.2	0.99	-

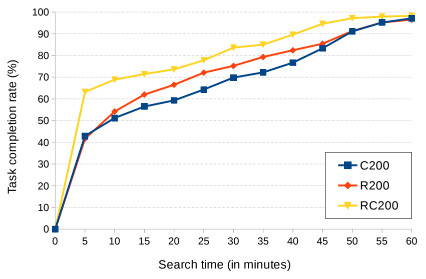
Table 1: Comparison of our MCTS solutions to the best known solutions (found by 16 different methods) on Solomon Benchmark

We observe that our approach in general is more successful for test instances where the number of robots is large and time windows are not very large. For example, the algorithm struggles to fully allocate tasks in C2 test instances, where the number of robots is 3 for all instances, while it does better in all type 1 instances where there are 9 or more robots. In type 2 instances each robot performs many more tasks. Having more robots enable random rollouts to complete more tasks. As shown in Figure 4, in 5 minutes we obtain much higher completion rates in type 1 scenarios. Also, we obtain best solutions for RC2 in type 2 instances as shown in Table 2. Our observation is also supported by the fact that in average RC2 has at least 50% more robots than other type 2 scenarios.

Detailed distance ratio results are reported in Table 1 and Table 2. The results reported in Table 1 show that our algorithm yields solutions with quality at most 1.59 away from the best-known for completed instances. These results are corroborated by the average distance ratio results in Table 2.

Similar to the completion results, our algorithm yields larger distance ratio values in type 2 data instances compared to type 1 ones. As stated before, the larger time windows cause the algorithm to evaluate more policies. Hence, we





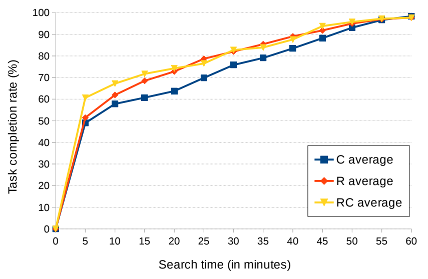


Figure 4: Task completion rates versus search time with MCTS for problems from the Solomon dataset. On average, MCTS achieves 50% task completion rate in 5 minutes. With more planning time, we achieve up to 98% completion rate over all categories.

	C1	R1	RC1	C2	R2	RC2	Avg.
Completion Rate Avg.	0.99	0.99	0.97	0.97	0.97	0.98	**0.98**
% Instances all completed	0.67	0.75	0.25	0.00	0.27	0.63	**0.43**
Ratio to Opt. Distance	1.39	1.25	1.28	-	1.51	1.48	**1.38**

Table 2: Summary of results for MCTS using the same number of robots of the best known solutions for Solomon benchmark.

Scenarios	C	R	RC	**All**
Distance Ratio	2.01	1.41	1.58	**1.67**
Team-size Ratio	2.65	2.11	1.92	**2.23**

Table 3: The ratio of found solutions to the best known are presented for MCTS with free number of robots within Solomon data set.

argue that larger run times would improve solution quality, given that the algorithm would be able to eventually focus the search away from allocations with larger distances.

4.3 Free number of robots

Given that it is challenging to complete all the tasks through random sampling with the tight robot team sizes we obtained from the best known solutions, we have experimented our approach by keeping everything the same but only changing the number of robots to be the same as the number of tasks to guarantee task completion with random rollouts. This setup also simplifies our evaluation function as $\psi = 0$ and $k = 1$ both hold. However, as expected, using more robots causes extra distance cost of leaving the depot and coming back.

We present a summary of the results obtained in this experiment in Table 3. This approach overall uses 123% more robots with an overall solution quality within 1.67 of the best known ones. Our first observation is that within each category as the time windows get tighter, MCTS finds solutions using more robots resulting in large team travel distances. Secondly, we obtain worst results for the clustered test cases as multiple robots are possibly assigned to the same clusters inflicting large distance costs, and lastly we obtain best results for the R-type scenarios, the one with no clusters.

4.4 Analysis

When the number of robots is fixed, the random trajectories during rollouts fail to accomplish all the tasks due to the highly constrained nature of the problem. Our evaluation function punishes task allocation policies with uncompleted tasks to direct the search towards regions where the robots are likely to complete more tasks with smaller distances.

Our approach performs better for problems where n is not very small. We think that this might be due to a weakness caused by our evaluation function and our tree structure model which uses robot r_{i+1} once robot r_i returns to the depot. Although our evaluation function punishes incomplete policies with an additional distance cost, the search can be biased towards individual robot routes with smaller distances for test instances with small n. As each robot route contains many tasks for small n, MCTS can only recognize late that it cannot generate a policy which completes all the tasks through a good looking branch.

In our approach even a single sub-optimal allocation made early in the plan diminishes the quality of a fully complete policy. The main challenge is that the delayed rewards for actions resulting from earlier allocations can be understood much later. Test cases with smaller n further increase the delay of acquiring less noisy rewards due to longer routes.

5 Conclusions

We proposed an MCTS based anytime centralized approach to solve the multi-robot task allocation problem with time windows and capacity constraints. The proposed MCTS heuristic combines branch and bound pruning and a parameterized root parallelization to obtain high quality solutions while maintaining relatively low computation times. We experimentally show that our approach can generate near-optimal task allocation policies in an hour using the Solomon benchmark for vehicle routing with 100 tasks. We found solutions that are at most 1.59 away from the best-known solutions, while completing nearly all tasks. Our method maintains asymptotic completeness guarantees of the MCTS algorithm as we employ no biasing or domain-dependent heuristic during the search.

Acknowledgments: Work supported in part by the University of Minnesota Informatics Institute and the Graduate School.

References

[Alighanbari *et al.*, 2003] M. Alighanbari, Y. Kuwata, and J. P. How. Coordination and control of multiple UAVs with timing constraints and loitering. In *Proc. American Control Conf.*, pages 5311–5316, June 2003.

[Auer *et al.*, 2002] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.

[Bräysy and Gendreau, 2005a] Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part I: Route construction and local search algorithms. *Transportation Science*, 39(1):104–118, February 2005.

[Bräysy and Gendreau, 2005b] Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part II: Metaheuristics. *Transportation Science*, 39(1):119–139, February 2005.

[Browne *et al.*, 2012] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, et al. A survey of Monte Carlo Tree Search methods. *IEEE Trans. on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[Chaslot *et al.*, 2008] Guillaume MJ-B Chaslot, Mark HM Winands, and H Jaap van Den Herik. Parallel Monte-Carlo Tree Search. In *Computers and Games*, pages 60–71. Springer, 2008.

[Coulom, 2006] Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In H. Jaap van den Herik, Paolo Ciancarini, and Jeroen Donkers, editors, *Computers and Games*, volume 4630 of LNCS, pages 72–83. Springer, 2006.

[Edelkamp and Gath, 2014] Stefan Edelkamp and Max Gath. Solving single vehicle pickup and delivery problems with time windows and capacity constraints using nested monte-carlo search. In *ICAART*, volume 1, pages 22–33, 2014.

[Enzenberger *et al.*, 2010] Markus Enzenberger, Martin Müller, Broderick Arneson, and Richard Segal. Fuego – an open-source framework for board games and Go engine based on Monte Carlo tree search. *IEEE Trans. on Computational Intelligence and AI in Games*, 2(4):259–270, 2010.

[Fern and Lewis, 2011] Alan Fern and Paul Lewis. Ensemble Monte-Carlo planning: An empirical study. In *Proc. Int'l Conf. on Automated Planning and Scheduling*, 2011.

[Galvan-Lopez *et al.*, 2014] Edgar Galvan-Lopez, Colin Harris, Leonardo Trujillo, Katya Rodriguez-Vazquez, Steven Clarke, and Vinny Cahill. Autonomous demand-side management system based on Monte Carlo tree search. In *Proc. IEEE Int'l Energy Conference*, pages 1263–1270, 2014.

[Gombolay *et al.*, 2013] Matthew Gombolay, Ronald Wilcox, and Julie Shah. Fast scheduling of multi-robot teams with temporospatial constraints. In *Robotics: Science and Systems*, 2013.

[Kartal *et al.*, 2014] Bilal Kartal, John Koenig, and Stephen J Guy. User-driven narrative variation in large story domains using Monte Carlo Tree Search. In *Proc. Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 69–76, 2014.

[Kartal *et al.*, 2015] Bilal Kartal, Julio Godoy, Ioannis Karamouzas, and Stephen J Guy. Stochastic tree search with useful cycles for patrolling problems. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 1289–1294, 2015.

[Kartal *et al.*, 2016] Bilal Kartal, Nick Sohre, and Stephen Guy. Generating sokoban puzzle game levels with monte carlo tree search. In *The IJCAI-16 Workshop on General Game Playing*, page 47, 2016.

[Kocsis and Szepesvári, 2006] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *Proc. European Conf. on Machine Learning (ECML)*, pages 282–293. Springer, 2006.

[Koes *et al.*, 2005] Mary Koes, Illah R. Nourbakhsh, and Katia P. Sycara. Heterogeneous multirobot coordination with spatial and temporal constraints. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 1292–1297, 2005.

[Korsah *et al.*, 2010] G. Ayorkor Korsah, Anthony Stentz, M. Bernardine Dias, and Imran Aslam Fanaswala. Optimal vehicle routing and scheduling with precedence constraints and location choice. In *Workshop on Intelligent Autonomous Systems at IEEE ICRA*, 2010.

[Korsah *et al.*, 2013] G. Ayorkor Korsah, Anthony Stentz, and M. Bernardine Dias. A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research*, 32(12):1495–1512, 2013.

[Lisỳ *et al.*, 2015] Viliam Lisỳ, Marc Lanctot, and Michael Bowling. Online Monte Carlo counterfactual regret minimization for search in imperfect information games. In *Proc. Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 27–36, 2015.

[McIntire *et al.*, 2016] Mitchell McIntire, Ernesto Nunes Nunes, and Maria Gini. Iterative auction for non-overlapping task scheduling. In *Proc. Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, 2016.

[Nam and Shell, 2015] Changjoo Nam and Dylan Shell. Assignment algorithms for modeling resource contention in multirobot task allocation. *IEEE Trans. on Automation Science and Engineering*, 12(3):889–900, 2015.

[Nunes and Gini, 2015] Ernesto Nunes and Maria Gini. Multirobot auctions for allocation of tasks with temporal constraints. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 2110–2116, 2015.

[Özkil *et al.*, 2009] Ali Gürcan Özkil, Zhun Fan, Steen Dawids, H Aanes, et al. Service robots for hospitals: A case study of transportation tasks in a hospital. In *Proc. IEEE Int'l Conf. on Automation and Logistics*, pages 289–294, 2009.

[Ponda *et al.*, 2010] S. S. Ponda, J. Redding, Han-Lim Choi, J.P. How, M. Vavrina, and J. Vian. Decentralized planning for complex missions with dynamic communication constraints. In *Proc. American Control Conf.*, pages 3998–4003, 2010.

[Portugal *et al.*, 2015] David Portugal, Paulo Alvito, Jorge Dias, George Samaras, Eleni Christodoulou, et al. Socialrobot: An interactive mobile robot for elderly home care. In *2015 IEEE/SICE Int'l Symposium on System Integration*, pages 811–816, 2015.

[Silver *et al.*, 2016] David Silver, Aja Huang, Chris J Maddison, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[Solomon, 1987] Marius M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.

[Solomon, 2005] Marius M. Solomon. VRPTW benchmark problems. `http://web.cba.neu.edu/~msolomon/problems.htm`, 2005.

[Veloso *et al.*, 2012] Manuela Veloso, Joydeep Biswas, Brian Coltin, Stephanie Rosenthal, Tom Kollar, Cetin Mericli, et al. Cobots: Collaborative robots servicing multi-floor buildings. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 5446–5447, 2012.