Task allocation for spatially and temporally distributed tasks

Julio Godoy and Maria Gini

Abstract In multi robot task allocation, a set of tasks has to be allocated to a group of robots while optimizing some measure (for example, fuel or time). In order to find the optimal allocation, an exponential number of possibilities must be explored. In this work, we extend the Consensus Based Bundle Algorithm, to improve its support for tasks with time constraints. The modified algorithm is compared with the original one in order to show how strategic modifications to the algorithm increase the number of tasks successfully completed.

1 Description of Problem

Multi robot task allocation addresses the problem of matching tasks to robots in a way that satisfies a certain criterion or optimization function. The criterion is usually to minimize the distance traveled, or to minimize the time to complete all the tasks, or maximize the number of completed tasks. Real problems often have additional constraints. For example, in a post-disaster scenario, tasks location and their magnitude may be initially unknown, so robots should first explore the environment and dynamically allocate tasks as they find them. Also, the amount of time to reach tasks might be limited, after which some of them may no longer be available (for example, a wounded civilian may have died), or may be too complex to complete (for example, a fire may be too big to put off). There might be obstacles in the predicted path to a task, and some tasks may need a minimum number of robots allocated in order to be completed.

We build upon a decentralized auction algorithm, the Consensus Based Bundle Algorithm [1], and modify it in order to incorporate stronger temporal constraints. This is useful for scenarios where, for example, the lives of wounded civilians depend on how fast ambulances can reach them.

Department of Computer Science and Engineering, University of Minnesota, 200 Union St SE, Minneapolis, MN 55455, USA e-mail: godoy@cs.umn.edu,gini@cs.umn.edu

2 Related Work

The area of multi robot task allocation has seen many contributions over the last decade. In this work, we focus on allocation methods based on auctions. A comparison of auction-based methods and token-based approaches [11] shows that auctions produce higher rewards but require more communications. However, Kalra et al. [4] compared both approaches and concluded that, when the task information is accurate, auction based methods achieve better performance.

When using auctions to allocate tasks, agents submit bids for the desired tasks. The amount of each bid is the value of the task for the agent, or is related to an objective metric, such as the distance. The auctioneer selects the agent which submitted the best bid for each task, and assigns it to that particular agent. Since we assume the agents are cooperative, social welfare increases when the most valuable combination of tasks is assigned to each agent.

Sandholm [9] proposes an algorithm for combinatorial auctions, a method that returns the optimal solution, but of exponential complexity, which makes it unfeasible for use in multi robot domains. Dias et al, [2] first discussed the use of auctions in the context of task allocation in a multi robot system, while Koenig et al. [5] proposed an efficient auction based algorithm which allows robots to account for already assigned tasks in their subsequent bids.

Auctions are distributed methods, but require communication among all the robots. To avoid the complete communication requirement, several approaches use consensus based algorithms [12, 1], where each robot determines independently its tasks and an equilibrium is reached by iteratively sharing information with its local neighbors.

When tasks have time constraints, the allocation algorithms have to handle scheduling in addition to cost. In [3] sequential auctions are used for situations where the reward for completing tasks decreases over time. In this case, time windows are not allowed to overlap. Similarly in [6] each task has a specific time window but again with no overlap. Overlapping time windows are not allowed as they would violate the constraint that there is a strict total order of the tasks. Finally, Ponda et al. [7] proposed a modification of the CBBA algorithm that deals with soft time constraints: agents have to arrive to a task before its deadline, without considering the time it takes the agent to complete the task. Ramchurn et al. [8] uses mixed integer programming to solve the problem of finding coalitions to do tasks, considering distance (allocating tasks to closer agents) and time (to allocate tasks with deadlines to agents that can reach them and execute them before the deadline).

In this work we consider tasks that have overlapping time windows and whose duration can be shorter than the time window. In our approach, time windows are considered hard constraints, so each task has to be completed within its time window.

3 Tasks with Time Windows

In many real world scenarios, tasks are not always available. For example, in rescue scenarios, there might be wounded civilians that could perish if not assisted on time. They might also be buried under semi collapsed buildings, that may collapse completely after some time, preventing them from being rescued afterwards. Fires might appear, and they might become un-extinguishable if they grow loo large.

These urgent tasks may appear anytime, or may all appear at the same time. It becomes critical that an agent not only reaches a task, but also that it reaches it in a specific time period, where the task is doable. This time period is called *time window*. Due to these extra constraints, tasks now have time related properties associated with them, specifically:

- *Start time* is the time from when the task can be done, the beginning of its time window. Before this time, the task does not exist.
- *End time* is the last time point an agent can work on the task. It marks the end of the time window of the task. After this time, the task does no longer exist.
- *Duration* is the time it takes an agent to do the task, once it is in the task location. Its length is at most the length of the task's time window.

A task is doable if the agent can reach it no later than $(Endtime - Duration)$, as otherwise it cannot complete it before its end time. For simplicity, we consider only tasks that are done by a single agent, so duration is a fixed number for each task. We also assume that task locations are known, and that there are no obstacles in the paths.

4 Extensions to CBBA

Our work extends the CBBA algorithm [1] and its modification in [7] where each task has an associated time window. We start by summarizing CBBA. The CBBA algorithm follows a two phase design:

Phase 1: Bundle Construction

Each agent assigns a score to each task (based on time or spatial criterion) and one by one selects the task with the maximum score, amongst the unassigned ones. This is repeated until all tasks have been assigned or the maximum bundle size has been reached. This is different from traditional bundle algorithms [9], where all possible bundle combinations are tested.

An agent has two lists of tasks, the bundle itself (b_i) and the path (p_i), which contains the same tasks as the bundle, but in the order they will be visited. Let L_t be the maximum bundle size and $S_i^{p_i}$ the total reward agent i receives by doing the tasks on path p_i. Each task is inserted in a position in the agent's path p_i where it maximizes the score improvement. $\oplus_n$ is the operation of inserting the second list after the nth element of the first list, while $\bigoplus_{end}$ appends the second list to the first.

Algorithm 1 CBBA Phase 1 for agent i at iteration t (from [1])

procedure Build Bundle($z_i(t\text{-}1)$,$y_i(t\text{-}1)$, $b_i(t\text{-}1)$)
$y_i(t)$=$y_i(t\text{-}1)$; $z_i(t)$=$z_i(t\text{-}1)$; $b_i(t)$=$b_i(t\text{-}1)$; $p_i(t)$=$p_i(t\text{-}1)$
while $|b_i| < L_t$ **do**
  $c_{ij} = \max_{n \leq |p_i|} S_i^{p_i \oplus_n \{j\}} - S_i^{p_i}, \forall j \in J / b_i$
  $h_{ij} = \mathbb{1}(c_{ij} > y_{ij}), \forall j \in J$
  $J_i = \arg\max_j c_{ij} \cdot h_{ij}$
  $n_{i,J_i} = \arg\max_n S_i^{p_i \oplus_n J_i}$
  $b_i = b_i \bigoplus_{end} J_i$
  $p_i = p_i \bigoplus_{n_{i,J_i}} J_i$
  $y_{i,J_i}(t) = c_{i,J_i}$
  $z_{i,J_i}(t) = i$
end while
end procedure

Algorithm 1 [1] summarizes the first phase of CBBA, where y_i is the winning bid list, z_i the winning agent list, and b_i and p_i respectively the bundle and path lists. J is the set of all possible combinations of tasks, of which each bundle is an instance. $\mathbb{1}(.)$ is the indicator function (= 1 if argument is true and 0 otherwise),

Phase 2: Conflict Resolution

After each agent has a bundle of pre allocated tasks, the agents communicate with each other, comparing their bids for the tasks in their bundles with other agent's bids for the same tasks. When an agent is outbid by another for a task, not only it has to release that task, but also all others it was planning to visit after that, as the score associated with them is no longer valid. In order to reach consensus, three lists are shared amongst the agents: the winning bids list y_i, the winning agent list z_i and a vector of time stamps of thelast information update from other agents, s_i.

At the time agent i receives a message from another agent k, using the information of lists z_i and s_i, it can take one of three actions [1] on task j:

1. update: $y_i j = y_k j, z_i j = z_k j$
2. reset: $y_i j = 0, z_i j = \emptyset$
3. leave: $y_i j = y_i j, z_i j = z_i j$

For example, an update action is taken when a better bid is found, a reset action is performed when both agents i and k believe that each other is the winner of task j and, in case both agents agree on the winner, no action is performed. An interesting feature of this phase is that the agents don't need to communicate directly with each other to reach consensus, but only to form a connected graph. This puts less constraints on the communication, and makes the algorithm suitable for environment conditions where communication is restricted. Detailed cases and the respective actions can be found in [1].

4.1 Maximizing the number of completed tasks

In the implementation of CBBA the task start and end times, agents, and tasks positions are randomly generated. The task duration is equal to the length of its time window. The implementation leaves room for improvement. First, it doesn't consider the ending time of tasks in the score function that assigns a value for each agent to perform a task. Hence, two tasks with the same starting time but different ending time (one more urgent than the other), are considered equal in terms of value for the agent. Second, a task is considered completed even if the task duration exceeds its ending time.

We modified the score function and evaluation mechanism to maximize the number of completed tasks, creating a variant of CBBA that we call MOD, and compared the results to the normal CBBA implementation. We consider an agent able to complete a task only if it can reach the task in a time that allows for its total completion (the duration of the task) before its time window ends.

Specifically, we consider both starting and ending times of tasks when computing the scores. Also, we added a sub algorithm to CBBA in order to test the feasibility of adding a task in the time before other already assigned tasks. This situation may arise when, after allocating the most valuable tasks, the algorithm finds that it can add another task (which previously was not very valuable), but because of time constraints, the new task has to be done before other tasks already allocated. This may create a situation where a previously assigned task is no longer doable in its time window, or needs to be shifted inside its time window.

We implemented a two way check to ensure that when a task is inserted before another, no time constraint is violated. The first check is via a value called *MinAlloTime*, which keeps track of the task with the smallest difference between the ending time and the current completion time in the temporal allocation. This way, if the new task has a duration longer than *MinAlloTime*, it is discarded and the insertion is aborted. The second check is via a procedure shown in Algorithm 2, in which the time added for inserting a new task m in position j is compared with the time constraints of all tasks that would be done after it in the agent bundle. If the arrival time of the agent to task $j+1$ after doing m is altered, then a check is done to ensure that task in position $j+1$ can still be done before the end of its time window. If not, the insertion is not feasible in position j and the next position $j+1$ is checked. Otherwise, the task is inserted in position j and the starting time for task in position $j+1$ is updated (i.e. the task is shifted) to reflect this insertion. In this case, all consecutive tasks must also be checked in case they also need to be shifted inside their time windows.

In Algorithm 2, a task is added before others if (1) the agent has empty slots in its bundle, (2) the task is the most valuable amongst the remaining ones; and (3) tasks already allocated can still be done in their respective time windows.

The difference is noticeable in the simple examples in Figure 1 and Figure 2. CBBA doesn't allocate task 2 to the agent (there is a single agent in this example), even though there is enough time to complete it. Due to the way the score function is designed, the agent just performs task 1 and 3.

Algorithm 2 Feasibility determination for agent i to do task m in position j of path p_i

$AddedT = start_m + duration_m + TravelT(m, Task_{j+1})$
if $AddedT < start_{Task_{j+1}}$ **then**
  return False
else
  if $AddedT > start_{Task_{j+1}}$ **then**
    if $AddedT + duration_{Task_{j+1}} < end_{Task_{j+1}}$ **then**
      $start_{Task_{j+1}} = AddedT$
      return True /* Task m feasible at position j */
    else
      return False /* Task m unfeasible at position j */
    end if
  end if
end if

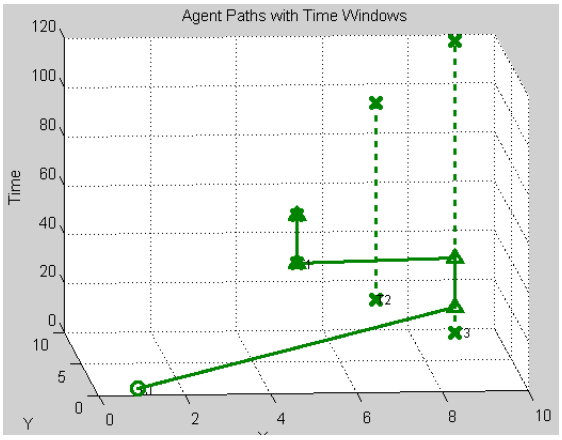

Fig. 1 Solution produced by CBBA

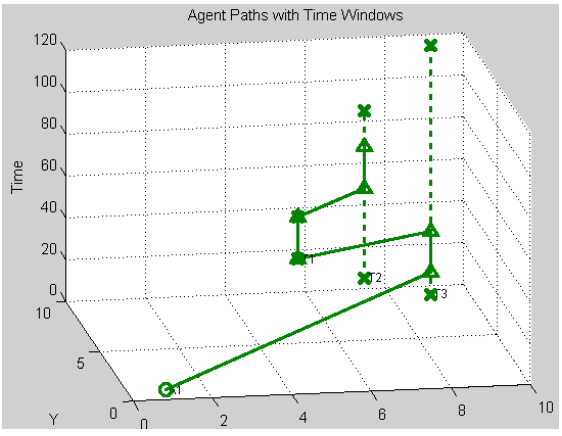

Fig. 2 Solution produced by MOD

5 Experimental Results

To test the impact of our modifications on the number of completed tasks, we tested MOD against the original CBBA in different settings, where the number of agents and tasks varied, as so did the tasks duration, length, start and ending time of their time windows. Finally, we also varied the size of the grid where the agents and tasks were placed. Because the grid is not made of discrete cells but of continuous X,Y coordinates, the increase of the size is the same as having the same grid but with the agents moving slower or faster.

The experiments considered using from two to ten agents, and from 10 to 100 tasks in different combinations, in order to evaluate specific scenarios. Each experimental setting was run 30 times, the resulting values are an average over the total set of results. The statistical significance of each experiment was tested with $p \leq 0.05$.

To start, we compare the number of tasks completed out of 10 in a scenario with two agents, with different task duration times. Table 1 summarizes the results.

Table 1 Two agents and 10 tasks with different task durations (TD)

Total number of tasks	TD=5	TD=10	TD=20	TD=40
CBBA	9.3333	8.3333	6.5333	4.3667
MOD	9.7333	9.3333	8.1667	5.7667

The differences in the results in Table 1 are statistically significant, except when task duration is 5. As task duration increases, the proportional difference in the average number of completed tasks between CBBA and MOD increases, favorably to MOD. At the same time, the average number of completed tasks decreases for all approaches, due to the higher chance of having short overlapping time windows.

The next experiment involves three agents and 50 tasks. Table 2 shows average number of tasks completed, total time to completion and total distance traveled by the agents. The values in Table 2 reflect the number of completed tasks. The approaches that complete more tasks also have a higher average path cost and larger average time needed to complete the tasks.

Table 3 shows the results for a larger scenario. This is useful to analyze the scalability of the proposed approach to longer distances. 10 agents are placed in a grid 10 times larger than the one used in previous experiments, with 100 tasks. In this case, MOD completes in average 50% more tasks than CBBA.

In order to analyze how both approaches perform under specific time window configurations, we include two additional scenarios: (a) the task duration is equal to its time window, and (b) the time window is four times the length of the task duration. For the first scenario, the difference between the two approaches is not statistically significant, mainly because the agents have fewer chances to vist the tasks, due to their constrained time windows.

Table 2 Three agents and 50 tasks, using multiple metrics with different task durations (TD)

Total number of tasks	TD=5	TD=10	TD=20	TD=40
CBBA	31.5333	21.5	13.3667	7.8667
MOD	30.3	24.3	16.6	10.3667

Total time	TD=5	TD=10	TD=20	TD=40
CBBA	114.8018	123.7246	136.1574	156.6719
MOD	153.0252	169.5668	186.7175	206.6135

Total distance	TD=5	TD=10	TD=20	TD=40
CBBA	95.4677	59.5234	33.5156	14.2594
MOD	86.693	63.1454	42.0853	23.3113

Table 3 Ten agents and 100 tasks, in a scenario ten times larger than the scenario used for the other experuments

Total number of tasks	TD=5	TD=10	TD=20	TD=40
CBBA	39.0333	32.8	25.4	18.5667
MOD	52.3667	48.1	38.0333	27.5667

Table 4 shows the number of completed tasks for scenario (b). In this case, because the agents have more room to decide when to complete a certain task, the modifications to CBBA improve performance.

Table 4 Three agents and 20 tasks, with time windows four times the task duration (TD)

Total number of tasks	TD=5	TD=10	TD=20	TD=40
CBBA	18.4	16.2	11.5333	7.1667
MOD	15.6667	16.6667	15.2667	12.9333

Table 5 shows the number of completed tasks for different agent-task configurations that have the same early start time, emulating a real-world scenario where a disaster occurs and suddenly many tasks appear that differ in both time duration and end time. It can be seen that MOD performs much better than the original CBBA, due to the change in the constraints and the task shifting in order to maximize the completion rate. In both cases, low values indicate overlapping time windows.

Finally, the number of iterations of the algorithm before reaching consensus was also tested, in experiments performed with different numbers of agents and tasks.

Table 5 Number of tasks completed when all tasks have the same start time

Total number of tasks	2A10T	3A20T	3A50T	10A100T
CBBA	2.8667	4.333	4.8	12.9333
MOD	6.2333	11.0714	17	31.5

The total number of iterations required by MOD is just 35% of the ones required by CBBA.

The modifications made to CBBA increase the number of completed tasks. Although with small tasks duration times the performance is in many cases similar to CBBA, as this time increases the enhancements start to show off, enhancing performance differences (in terms of number of completed tasks) that are most visible in the case where all tasks have the same start time. Increasing the number of completed tasks is not trivial, as it requires to find a tradeoff between space and time, and a detailed analysis of how each of these dimensions affects the possibility of completing more tasks is something that goes beyond the scope of this paper.

Another interesting improvement is in the number of iterations required to reach convergence, which is less than half as normal CBBA. This implies that by using MOD, the algorithm is less vulnerable to communication errors, and hence it is more adequate for use in communication constrained environments.

6 Conclusions and Future Work

We presented our modifications to CBBA in order to improve its performance with tasks that have time windows. We have presented how the tradeoff between space, time, and communication affects the number of tasks completed. Some results matched our initial intuition while others showed us some interesting and unforeseen correlations, that give light to future enhancements. The number of tasks completed increases to different degrees depending on the specific setting. The number of communication rounds required to reach consensus is also greatly reduced, opening new possible applications to the algorithm.

As future work, we will perform a theoretical analysis of our approach and aim at improving specific performance metrics while, at the same time, keeping an appropriate overall performance balance. For instance, we will consider adding levels of criticality to the tasks, so when not all the tasks can be done, the agents could work on the most critical tasks first. Another line of future research is related to inter-task constraints, similar to the approach developed in [10]. Specifically, we will analyze how precedence constraints between tasks and their decomposition affects performance and convergence time of the multi-robot system.

References

1. Choi, H.L., Brunet, L., How, J.P.: Consensus-based decentralized auctions for robust task allocation. IEEE Trans. on Robotics **25**, 912–926 (2009). DOI 10.1109/TRO.2009.2022423. URL http://dl.acm.org/citation.cfm?id=1653147.1653161
2. Dias, M.: Traderbots: A new paradigm for robust and efficient multirobot coordination in dynamic environments. Ph.D. thesis, Carnegie Mellon, Pittsburgh, Pennsylvania (2004)
3. Ekici, A., Keskinocak, P., Koenig, S.: Multi-robot routing with linear decreasing rewards over time. In: Proc. Int'l Conf. on Robotics and Automation, pp. 3944–3949 (2009). URL http://dl.acm.org/citation.cfm?id=1703775.1704082
4. Kalra, N., Martinoli, A.: Comparative study of market-based and threshold-based task allocation. In: M. Gini, R. Voyles (eds.) Distributed Autonomous Robotic Systems 7, pp. 91–101. Springer Japan (2006)
5. Koenig, S., Tovey, C., Lagoudakis, M., Markakis, V., Kempe, D., Keskinocak, P., Kleywegt, A., Meyerson, A., Jain, S.: The power of sequential single-item auctions for agent coordination. In: American Association of Artificial Intelligence, pp. 1625–1629 (2006). URL http://dl.acm.org/citation.cfm?id=1597348.1597457
6. Melvin, J., Keskinocak, P., Koenig, S., Tovey, C.A., Ozkaya, B.Y.: Multi-robot routing with rewards and disjoint time windows. In: Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, pp. 2332–2337 (2007)
7. Ponda, S.J., Redding, J., Choi, H.L., How, J.P., Vavrina, M.A., Vian, J.: Decentralized planning for complex missions with dynamic communication constraints. In: American Control Conference (2010)
8. Ramchurn, S.D., Polukarov, M., Farinelli, A., Truong, C., Jennings, N.R.: Coalition formation with spatial and temporal constraints. In: Int'l Conf. on Autonomous Agents and Multi-Agent Systems, pp. 1181–1188 (2010)
9. Sandholm, T.: Algorithm for optimal winner determination in combinatorial auctions. Artif. Intell. **135**, 1–54 (2002). DOI 10.1016/S0004-3702(01)00159-X
10. Whitten, A.K., Choi, H.l., Johnson, L.B., How, J.P.: Decentralized Task Allocation with Coupled Constraints in Complex Missions, pp. 1642–1649. IEEE (2011)
11. Xu, Y., Scerri, P., Sycara, K., Lewis, M.: Comparing market and token-based coordination. In: Int'l Conf. on Autonomous Agents and Multi-Agent Systems, pp. 1113–1115. ACM, New York, NY, USA (2006)
12. Zavlanos, M.M., Spesivtsev, L., Pappas, G.J.: A distributed auction algorithm for the assignment problem. In: Proc. 47th IEEE Conference on Decision and Control, pp. 1212–1217 (2008)