Optimal Multi-Agent Multi-Task Planning

Hannah Lee, James Motes, Nancy M. Amato***

[1] Department of Computer Science,
Colorado School of Mines, Golden, CO, USA,
hannahlee@mines.edu.
[2] Parasol Lab, Department of Computer Science and Engineering,
University of Illinois Urbana-Champaign, Champaign, IL, USA,
{jmotes2, namato}@illinois.edu.

Abstract. In this paper, we present Task Conflict-Based Search (TCBS), an optimal multi-agent multi-task planning algorithm. This is an adaptation of the multi-agent pathfinding algorithm Conflict-Based Search (CBS) to task space. An individual task plan consists of a sequence of agent assignments transitioning the system from a start state to a goal state. The objective for multi-agent multi-task problems is to find plans for all tasks while avoiding agent allocation conflicts. TCBS performs a high-level search on a Conflict Tree (CT) where conflicts representing overlapping assignments of agents between tasks are converted into task constraints. Each node in the CT corresponds to a low-level search that produces a multi-task plan with respect to the task constraints found in the CT.

1 Introduction

Optimal multi-agent task allocation is shown to be an NP-hard problem (Turner). The problem becomes increasingly more complex when there are heterogeneous agents with different capabilities and constraints that are required to perform various subtasks. The objective of multi-agent multi-task allocation is to find an optimal team solution for an unordered set of independent motion tasks. Tasks can be completed in any order, using any available agents. Creating an optimal team solution is difficult because tasks are independently planned without taking into consideration shared and limited resources.

Multi-agent multi-task planning is useful for many applications such as manufacturing or unmanned aircraft vehicle coordination. The ability to allocate

* This research supported in part by NSF awards CNS-0551685, CCF 0702765, CCF-0833199, CCF-1439145, CCF-1423111, CCF-0830753, IIS-0916053, IIS-0917266, EFRI-1240483, RI-1217991, by NIH NCI R25 CA090301-11,and by DOE awards DE-AC02-06CH11357, DE-NA0002376, B575363.
** This work was performed at the Parasol Lab during Summer 2019 and supported in part by the CRA-W Distributed DREU project.

multiple tasks to multiple robots is useful in creating an undisturbed workflow where tasks are completed simultaneously. An individual task can also be split into subtasks where various robots sequentially take over ownership of the task to complete it efficiently. To find an optimal team solution, efficient and effective task and motion planning is required.

In this work, we present Task Conflict-Based Search (TCBS) to solve multi-agent multi-task planning (MAMTP) problems. We adapt Conflict-Based Search (CBS) [18], which was originally used to solve multi-agent path finding (MAPF) problems, for MAMTP. A solution for MAPF problems consists of a set of conflict-free paths for a given multi-agent team. Similarly, a solution for MAMTP problems consists of a set of conflict-free plans for the set of tasks. TCBS consists of two searches: a high-level search and a low-level search. Individual task plans are generated through a low-level search while a high-level search identifies and resolves conflicts between tasks.

The low-level search produces a task plan consisting of a sequence of individual agent subtasks and interactions. To build the search space, we use the combined roadmap from [13] to build an underlying roadmap for the multi-agent team with interactions. We abstract this graph into a `task graph` consisting of potential subtask start/end locations. This is integrated with agent availability w.r.t. time to provide a graph representation of possible task plans.

CBS defines conflicts as agents being at the same location at the same time. For MAMTP, conflicts represent overlapping assignment of agents between tasks. A conflict therefore consists of an agent and two incompatible agent assignments. These assignments are either directly overlapping in time or the assignment of the agent prevents other tasks from utilizing that agent at a future time.

The high-level search identifies conflicts and then converts them into a pair of constraints for the involved tasks. The existing solution is branched, and each constraint is passed to one of the child nodes. Exploration of this tree leads to resolutions of the individual task plan's conflicts and an optimal solution.

We first describe the state-of-the-art in MAMTP and the original CBS approach. Then, we detail the adaptation of CBS to TCBS and both the low-level and high-level search. Finally, we illustrate the effectiveness of the TCBS adaptation as well as adaptations of CBS variations to TCBS.

2 Related Work

2.1 Multi-Agent Task Planning

NP-Hard - Multi-agent task allocation has been extensively studied in the past. Finding an optimal-solution is strongly NP-hard [4,23,24]. MAMTP problems can be translated into a version of the traveling salesman problem where agents are represented by salesmen following paths instead of tours [4]. The key difficulty of multi-level optimization problems is that optimizing multi-agent allocation depends on solutions of a combinatorially explosive number of task assignments and conflicts, each of which is NP-hard and computationally expensive [24]. Because multi-agent task allocation is an NP-hard problem, many

multi-agent multi-task planners provide sub-optimal solutions when the number of tasks is large.

Optimal Planners Optimal multi-agent multi-task planners such as [11], [1], [12], and [10], use varied algorithms to return optimal solutions. However, because multi-agent task allocation is computationally expensive, these optimal planners are only reasonable and realistic to use for small numbers of tasks. Much previous work also involves speeding up previously presented algorithms so that they are more feasible to use in larger task problems. Optimal solutions tend to be computed using Branch-and-Bound, an algorithm that searches the state space of candidate solutions represented as a tree [8].

Sub-optimal Planners Sub-optimal multi-agent multi-task planners such as [19], [21], and [9], trade optimality for reduced execution times. Sub-optimal planners can typically find sub-optimal solutions very quickly. Sub-optimal planners focus on finding solutions efficiently while keeping the sub-optimal solution cost as close to the optimal solution cost. Sub-optimal planners are typically used for larger task problems due to significantly lower execution times.

Cooperation - Task planning requires task decomposition to break down an abstract task into subtasks based on factors such as environment and heterogeneous robots [7]. Each subtask is executed by a single robot. One approach to task decomposition is to use a centralized coordinator to decompose a task into subtasks [2, 6, 20]. This is the method we employ.

There are two commonly used methods for multi-agent group coordination: centralized and decentralized coordination. Centralized coordination requires all team members to communicate with a centralized coordinator which assigns tasks to each member within the team. Decentralized coordination requires team members to use distributed algorithms to coordinate and assign tasks amongst themselves [13]. In TCBS, we use a centralized coordinator for the task planning portion.

Centralized coordination is advantageous in that only a single agent is used to plan. This agent therefore plans using a global awareness of the system making it often reliable [16]. However, the downside of using centralized coordination is the necessity for constant and reliable communication channels between the coordinator and the team members [13].

Decentralized coordination methods, such as [16] and [15], are more responsive to uncertainty compared to centralized coordination. Using decentralized coordination, agents can modify their responses as they explore their environment. The failure of a single agent does not necessarily compromise the integrity of the entire system [13].

2.2 Multi-Robot Systems

Multi-robot systems (MRS) can be made of heterogeneous or homogeneous robots. MRS often provide significant advantage over utilizing a single robot. Heterogeneous systems have the ability to perform tasks that may require different hardware. Homogeneous systems can increase throughput by executing tasks in parallel. However, MRS requires detailed planning in order for a system to utilize each robot [13].

An important factor of MRS is the interaction of robots for the passing of a task from one robot to another. This can be required if a specific task either requires multiple disjoint robot types or if a task can be executed more efficiently with multiple robots of the same capability. In this work, we use the previously presented Interaction Template (IT) method for modeling multi-robot motion planning problems with interactions [13]. ITs are small roadmaps that define the interaction between two or more robots in an isolated setting. They are generated with standard motion planning algorithms as a pre-processing step and are tiled into the robots' individual roadmaps at appropriate locations to encode potential interactions. The ITs are then connected to form a combined roadmap that encodes possible paths for the robot team which satisfy the motion task.

2.3 Conflict-Based Search

Method - Conflict-Based Search is multi-agent pathfinding algorithm that searches a conflict tree (CT) to find a solution consisting of conflict free paths for all agents [18]. CT is a binary tree that contains a set of constraints, a solution, and the total cost of the current solution. The set of constraints starts as an empty set at the root of the CT. The child nodes in the CT inherits the constraints of its parent and adds a single new constraint for one agent. The solution is a set of paths where a path is planned under a set of specific constraints.

When a node within the CT is processed, a low-level search is invoked that returns the shortest path for each agent that is consistent with the given constraints. Once a path is found for each agent, the paths are then validated with respect to the other agents. If all agents reach their goal without any conflicts, the CT node is declared as a goal node and the current solution is returned. However, if a conflict occurs within the CT node, the validation halts and the node is declared a non-goal node.

A non-goal node is resolved by splitting the node into two children. Both children inherit the set of constraints from the node. They are passed the first conflict where two agents are occupying the same vertex at the same time. The first child resolves the conflict by adding a constraint of the second agents vertex and time to the first agent. Likewise, the second agent resolves the conflict by adding a constraint of the first agents vertex and time to the second agent. Once the CT search is completed, the minimum goal node holds the optimal path. CBS guarantees optimality by examining both possibilities as a node is split into two children.

Other Variations - There are many variations that have been studied to improve CBS. Some variations, such as Satisfiability Modulo Theories (SMT) CBS [22] and Meta-Agent Conflict-Based Search (MACBS) [17], improve the runtime of CBS while maintaining optimality. SMT-CBS formulates CBS using Satisfiability Modulo Theories to obtain optimal makespan solutions. MACBS couples groups of agents into meta-agents if the number of internal conflicts between agents exceeds a given bound.

Other variations trade optimality for dramatic improvements in runtime, such as Greedy-CBS (GCBS) and Bounded Suboptimal CBS (BCBS) [3]. GCBS uses the same framework as CBS but prefers to expand nodes that are more likely to quickly produce a valid, but possibly suboptimal, solution. BCBS applies CBS as a focal search.

Improved CBS (ICBS) is an improved version of the original CBS algorithm where optimality is conserved [5]. ICBS consists of two improvements: the merge and restart improvement, and the bypass improvement (BP).

MACBS involves merging agents into meta-agents [17]. This merge reduces the size of the CT at the cost of increased computational effort by the low-level solver (ICBS: Improved Conflict-based search). ICBS remedies this cost by discarding the current CT when a merge decision has been made and restarting the search from a new root node. The agents are merged into a meta-agent at the beginning of the new search.

BP attempts to prevent a split action and bypass the conflict by modifying the chosen path of one of the agents. A bypass to a path with respect to a conflict and a CT node is valid if the new path does not include the current conflict, if the cost of both paths are equivalent and if both paths are consistent with the CT nodes constraints. For a given node, BP peeks at either immediate child of the current node in the CT. If the path of a child is evaluated to include a helpful bypass to the current node, then the childs path is adopted by the current node without splitting the node and adding the child nodes to the CT. Although this is the same number of computations as CBS, it can potentially save a significant amount of search time and space due to the smaller CT size. We use BP to significantly speed up the high-level search in TCBS.

3 Method

3.1 Low-Level Search

In the initial MAPF CBS algorithm, the goal is to find a set of collision-free individual agent paths. These individual paths are computed with a low-level path finding algorithm [18]. The authors use A* over `a two dimension grid x time` search space in the original work for this. We map this concept to task space by creating a search space of `potential subtask start/end locations x agents x available intervals`. A path in this task space thus consists of a sequence of subtasks with corresponding agent assignments.

For determining possible subtask start/end locations, we utilize the combined roadmap concept from the Interaction Template (IT) method [13]. The IT instances in this combined roadmap represent exchanges in task ownership and thus the end of one subtask and the start of the next. We abstract this graph into a *task graph* and use the IT locations as vertices and edges between these vertices correspond to paths between them in the combined roadmap. The virtual start and goal vertices used in the IT method are carried over to the task graph as well. A plan across the task graph will produce a task plan for an individual task without considering agent availability. This process can be seen in Figures 1 and 2.

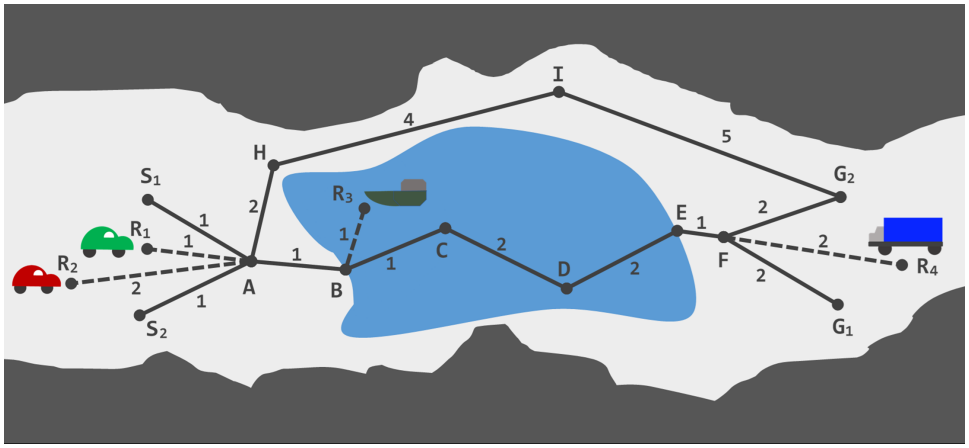

Fig. 1. Example of an environment's combined roadmap

We define an agent *available interval* as a contiguous period of time in which the agent is not occupied performing a subtask or traveling to or from the subtask start or end location respectively. This is the equivalent to a space being unoccupied by any agent for a contiguous period of time in the MAPF problem. As the task graph corresponds to the underlying combined roadmap which exists in a continuous environment, the time intervals for subtasks on this graph are also continuous. Mapping the original time dimension in the CBS work designed for a discrete space would thus result in a massive search space w.r.t agent availability. A continuous time extension to CBS proposed in [22] uses the Safe Interval Path Planning technique [14] to condense the time dimension of the search space. We adapt this concept to model agent availability with available intervals.

Each vertex in the task graph corresponds to a robot-type as described in [13]. The low-level search spaces consists of each of these locations by each available interval of all the agents of the corresponding type. The available intervals at a vertex for an agent are found by taking in a set of constraints representing

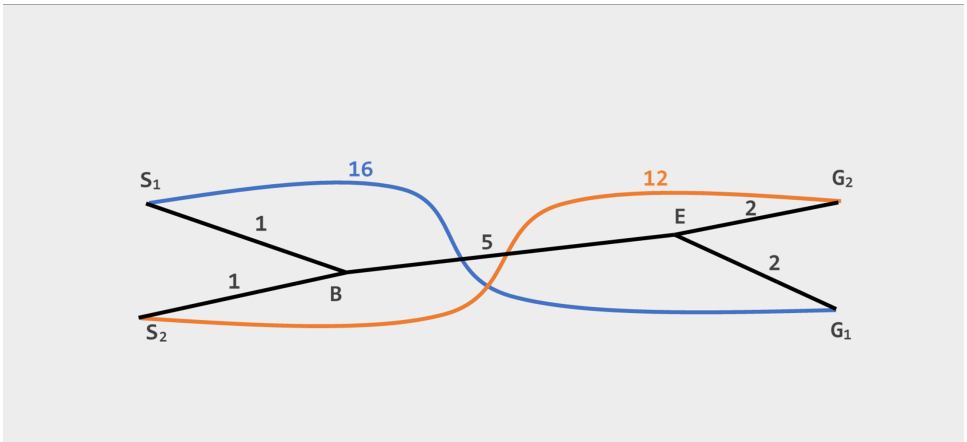

Fig. 2. Edges in task graph correspond to paths in combined roadmap (ex: $S_1 \rightarrow G_1$, distance = 16, path in combined roadmap: $S_1 \rightarrow A \rightarrow H \rightarrow I \rightarrow G_2 \rightarrow F \rightarrow G_1$)

existing subtask assignments of the agent and computing the time taken to travel to, perform, and return from the subtask. A search over this space produces an optimal individual task plan w.r.t. the existing assignment constraints of the agents.

3.2 High-Level Search

The high-level search algorithm builds a CT tree to iteratively find an optimal plan for a given MAMTP problem (Alg. 1). Nodes in the CT consist of multi-agent multi-task solutions and individual task constraints. These constraints are used to resolve conflicts between individual task plans.

Task conflicts consist of an agent, an occupied time interval, and start and end locations. These are constraints placed on tasks to prevent conflicts in agent allocation. A conflict occurs when an agent is assigned to multiple tasks during the same time or when an agents assignment to multiple tasks is not feasible due to time and location constraints. If a task occupies an agent for a given time interval and a different task occupies the same agent during a disjoint time interval, a conflict can still occur if the agent cannot reach the second task after completing the first task.

We define an occupied interval as a contiguous period of time during which an agent is completing a subtask. An occupied interval consists of a start and end time, a start and end configuration or location, and an agent. In TCBS, occupied intervals are used to define available and unavailable intervals. These are adaptations of the safe and collision intervals used in [14]. Available intervals are occupied intervals in which a tasks allocation of an agent does not conflict with other allocations of the same agent by other tasks. An unavailable interval is the opposite of an available interval. Unavailable intervals are occupied intervals

in which a tasks allocation of an agent conflicts with other allocations of the same agent. Conflicts may arise due to temporal constraints. Available intervals are used to create valid, optimal plans where the allocation of tasks to agents is conflict-free. Unavailable intervals are used to create conflict maps that place agent constraints on task planning.

The initial plan is constructed by having each task planned independently of each other using the low-level search. This initial plan is placed as the root of the Task Conflict Tree (TCT). After which, we begin looping and iterating through the tree until the search is completed. Within the looping sequence, we get the minimum node within the TCT and check each plan for task conflicts.

If a task conflict is found, it is added to a conflict map that places temporal agent assignment constraints on future plans. For a given task conflict, its respective tasks are then replanned and saved as child nodes of its parents. Thus, if a task conflict occurs between task 1 and task 2, the parent plan is replanned for task 1 in one of the child nodes and replanned for task 2 in the other child node while all other task plans in the solution are held constant.

If a task conflict is not found within a given node, the node is marked as a goal node and search along that branch is terminated. Once a goal node has been found, search along non-goal nodes can be terminated before they become goal nodes if and only if the solution cost of the non-goal node exceeds that of a goal node. As constraints are placed on the nodes of the TCT, the solution cost can only increase as the depth of the tree increases. Thus, once a non-goal node's cost exceeds that of a goal node's cost, all potential plans along that branch will not be the optimal solution. Once the search is completed, the minimum cost goal node will hold the optimal solution. Figure 3 shows the TCT that was generated from the combined roadmap (Fig. 1) and the low-level graph (Fig. 2).

TCBS can be used with either a sum of costs or makespan cost metric. Sum of costs is the sum of each plan's end time. Makespan is the maximum end time found within a set of task plans. Using sum of costs will return the shortest overall solution. Using makespan will return the shortest overall solution where each task will be complete as quickly as possible. The cost metric can affect the size of the TCT, but, if given enough time to complete its search, TCBS will return an optimal solution.

3.3 Improved Task Conflict-Based Search

After an initial set of experiments, we decided that TCBS was too inefficient to solve MAMTP problems. To remedy this, we applied BP [5] to TCBS to create Improved Task Conflict-Based Search (ITCBS). As stated previously, BP tries to prevent the splitting of parent nodes into child nodes by bypassing the conflict by modifying the chosen plan of one of the tasks. ITCBS follows the same algorithm as ICBS's BP improvement.

When a conflict is encountered, ITCBS peeks at either immediate child of the current node within the TCT. If the cost of both paths are equivalent and consistent with the current node's constraints, then the bypass is considered valid. Given a valid bypass, the child's path is adopted by the current node

Algorithm 1 Task Conflict-Based Search The high-level search takes in a set of agents and unordered motion tasks and returns a team solution consisting of valid plans for all the input tasks.

Input: set of tasks T, set of agents A
Output: final team solution S_f

```
S_f ← ∅ // infinite solution cost
for all task t ∈ T do
    S_0 ← LowLevelSearch(t)
root.solution ← S_0
root.cost ←Makespan(S_0)
TCT tree ← ∅
tree.insert(root)
while tree not empty do
    minNode ←tree.GetMinNode()
    if minNode has no conflict then
        if minNode.cost < Makespan(S_f) then
            S_f ← minNode.solution
    else
        if minNode.cost < Makespan(S_f) then
            C ← first conflict (t_i,int_i,t_j,int_j,a_k)
            for all task t ∈ C do
                N ← new node
                N.constraints←minNode.constraints + (t_i,int_j,a_k)
                N.solution update with LowLevelSearch(t_i)
                N.cost←Makespan(N.solution)
                tree.Insert(N)
return S_f
```

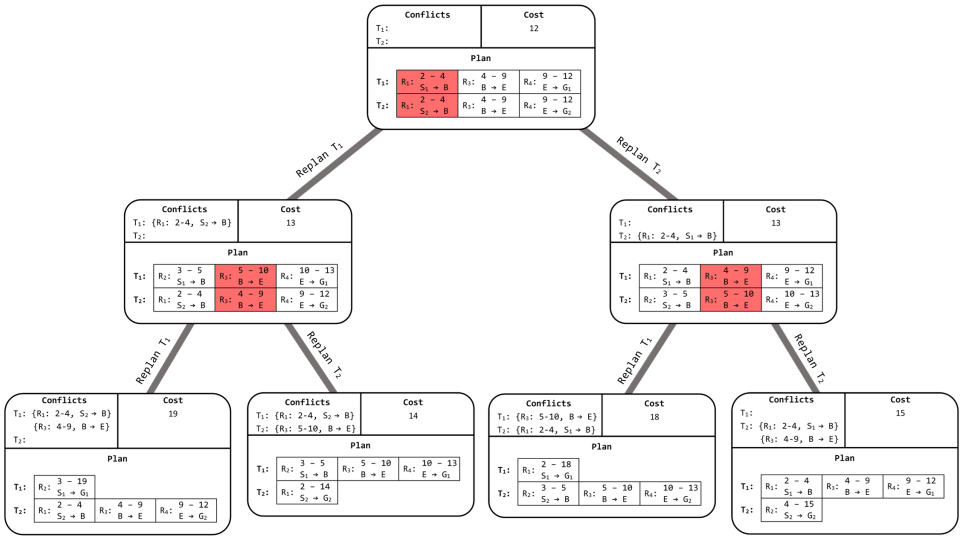

Fig. 3. Task Conflict Tree calculated from Figure 1

without splitting the node. Thus, the child nodes are not added to the TCT. This requires the same number of computations as TCBS; however, due to the smaller TCT size, the evaluation time becomes significantly lower as the number of tasks increase due to the fewer number of parent to child splits.

4 Experiments

4.1 Experimental Setup

The TCBS algorithm was tested using a scenario that involved a team of four heterogeneous agents: three ground robots and one water robot. These agents solved varying numbers of randomly generated tasks. The environment and starting locations of each agent was held constant throughout the experiments; however, the start and goal locations for each trial's tasks' were randomly generated. We ran a set of thirty trials using varying seeds to randomly generate these tasks using a makespan cost metric.

The environment used, as seen in Figure1, is comprised of a land mass with a single water mass in the center. There are three ground robots. Two of these robots have start locations on the left-hand side of the environment. These locations are relatively close to each other, increasing the number of conflicts. The third agent is on the far right of the environment. The ground robots can also interact with the water robot at vertices B and E.

We ran these experiments using both TCBS and ITCBS. By using the same seeds when running these search algorithms, we were able to provide the same randomly generated tasks to each algorithm. This allowed us to compare ITCBS to TCBS using the same set of task plans.

4.2 Results

When given random start and end locations for varying numbers of tasks, the size of the TCT varies greatly depending on the number of tasks and the complexity of the tasks. Given that there are four agents, when there are less than four tasks, it is significantly easier to solve the MAMTP problem. However, once the number of agents is equivalent to the number of tasks, the number of conflicts increases. This makes both the tree size and evaluation time increase exponentially as the number of tasks increases.

As seen in Tables 1 and 2, all metrics increase as the number of tasks approaches the number of agents. The implementation of ITCBS significantly improves the TCT size and the evaluation time as the number of tasks increase. We stopped running TCBS after five tasks due to the amount of time it took to solve these MAMTP problems. When examining the ITCBS results (Table 2), the evaluation time for five tasks was significantly higher than for that of six tasks. We suspect that this is due to a seed that produced a particularly difficult set of task plans to solve. However, despite this, the improvement in both tree size and evaluation time can be clearly seen between the two search algorithms.

Number of Tasks		Total Nodes	Minimum Node Depth	Maximum Node Depth	Nodes Explored	Evaluation Time (s)
1	avg:	1.00	0.00	0.00	1.00	0.06
	stdev:	0.00	0.00	0.00	0.00	0.01
2	avg:	2.83	0.83	0.83	2.8	0.10
	stdev:	1.58	0.57	0.57	1.57	0.03
3	avg:	8.74	2.09	2.29	7.26	0.24
	stdev:	11.43	1.44	1.74	9.02	0.21
4	avg:	107.03	4.74	6.91	90.54	3.96
	stdev:	161.12	2.05	3.81	140.99	7.66
5	avg:	927.12	8.94	14.74	726.47	646.27
	stdev:	1317.72	2.93	6.39	1064	1992.43

Table 1. TCBS Results using Four Agents

5 Conclusion

We have shown an optimal multi-agent task allocation algorithm for solving multi-agent multi-task planning problems. This method uses the previous developed [18], used to solve multi-agent pathfinding problems, and applies it to task space. As an NP-hard problem, the state space for multi-agent multi-task planning problems increases exponentially as the number of tasks increase. Due to the inefficiency of TCBS, we implemented ITCBS from [5], which significantly improved the evaluation time. Future work includes applying CBS variations to task space and seeing when sub-optimal planners may be preferred over optimal planners.

Number of Tasks		Total Nodes	Minimum Node Depth	Maximum Node Depth	Nodes Explored	Evaluation Time (s)
1	avg:	1.00	0.00	0.00	1.00	0.06
	stdev:	0.00	0.00	0.00	0.00	0.01
2	avg:	2.09	0.74	0.74	1.91	0.10
	stdev:	1.29	0.61	0.61	0.78	0.03
3	avg:	6.6	1.71	1.83	4.37	0.20
	stdev:	9.12	1.53	1.64	4.64	0.17
4	avg:	77.49	4.34	5.97	40	2.64
	stdev:	122.02	2.13	3.49	60.97	5.63
5	avg:	231.09	66.00	8.74	116.86	65.74
	stdev:	496.64	2.97	4.99	248.27	340.67
6	avg:	77.49	4.34	5.97	40.00	2.65
	stdev:	122.02	2.13	3.49	60.97	5.63

Table 2. ITCBS Results using Four Agents

References

1. Amador, S., Okamoto, S., Zivan, R.: Dynamic multi-agent task allocation with spatial and temporal constraints. In: Twenty-Eighth AAAI Conference on Artificial Intelligence (2014)
2. Aylett, R.S., Barnes, D.P.: A multi-robot architecture for planetary rovers. In: Proc. ESA Wksp. on Advanced Space Technologies for Robotics and Automation (1998)
3. Barer, M., Sharon, G., Stern, R., Felner, A.: Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In: Seventh Annual Symposium on Combinatorial Search (2014)
4. Bhattacharya, S., Likhachev, M., Kumar, V.: Multi-agent path planning with multiple tasks and distance constraints. In: Proc. IEEE Int. Conf. Robot. Autom. (ICRA). IEEE (2010)
5. Boyarski, E., Felner, A., Stern, R., Sharon, G., Tolpin, D., Betzalel, O., Shimony, E.: Icbs: improved conflict-based search algorithm for multi-agent pathfinding. In: Twenty-Fourth International Joint Conference on Artificial Intelligence (2015)
6. Caloud, P., Choi, W., Latombe, J.C., Pape, C.L., Yim, M.: Indoor automation with many mobile robots. In: IEEE International Workshop on Intelligent Robots and Systems, Towards a New Frontier of Applications (July 1990)
7. Chen, J., Yang, Y., Wei, L.: Research on the approach of task decomposition in soccer robot system. In: International Conference on Digital Manufacturing & Automation (December 2010)
8. Clausen, J.: Branch and bound algorithms-principles and examples. Department of Computer Science, University of Copenhagen pp. 1–30 (1999)
9. Gini, M.: Multi-robot allocation of tasks with temporal and ordering constraints. In: Thirty-First AAAI Conference on Artificial Intelligence (2017)
10. Jan, G.E., Chang, K.Y., Parberry, I.: Optimal path planning for mobile robot navigation. IEEE/ASME transactions on mechatronics 13(4), 451–460 (2008)
11. Lau, H.C., Zhang, L.: Task allocation via multi-agent coalition formation: Taxonomy, algorithms and complexity. In: Proceedings. 15th IEEE International Conference on Tools with Artificial Intelligence. pp. 346–350. IEEE (2003)

12. Macarthur, K.S., Stranders, R., Ramchurn, S., Jennings, N.: A distributed anytime algorithm for dynamic task allocation in multi-agent systems. In: Twenty-Fifth AAAI Conference on Artificial Intelligence (2011)
13. Motes, J., Sandström, R., Adams, W., Ogunyale, T., Thomas, S., Amato, N.M.: Interaction templates for multi-agent systems. Tech. Rep. TR18-002, Texas A&M University (2018)
14. Phillips, M., Likhachev, M.: Sipp: Safe interval path planning for dynamic environments. In: 2011 IEEE International Conference on Robotics and Automation. pp. 5628–5635. IEEE (2011)
15. Roth, M., Simmons, R., Veloso, M.: Decentralized communication strategies for coordinated multi-agent policies. In: Multi-Robot Systems. From Swarms to Intelligent Automata Volume III, pp. 93–105. Springer (2005)
16. Sanchez, G., Latombe, J.C.: Using a prm planner to compare centralized and decoupled planning for multi-robot systems. In: Proc. IEEE Int. Conf. Robot. Autom. (ICRA). vol. 2, pp. 2112–2119 (2002)
17. Sharon, G., Stern, R., Felner, A., Sturtevant, N.R.: Meta-agent conflict-based search for optimal multi-agent path finding. SoCS 1, 39–40 (2012)
18. Sharon, G., Stern, R., Felner, A., Sturtevant, N.R.: Conflict-based search for optimal multi-agent pathfinding. Artificial Intelligence 219, 40–66 (2015)
19. Shehory, O., Kraus, S.: Methods for task allocation via agent coalition formation. Artificial intelligence 101(1-2), 165–200 (1998)
20. Simmons, R., Apfelbaum, D., Fox, D., Goldman, R.P., Haigh, K.Z., Muslineg, D.J., Pelican, M., Thrun, S.: Coordinated deployment of multiple, heterogeneous robots. In: Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS) (October 2000)
21. Strens, M., Windelinckx, N.: Combining planning with reinforcement learning for multi-robot task allocation. In: Adaptive Agents and Multi-Agent Systems II, pp. 260–274. Springer (2004)
22. Surynek, P.: Multi-agent path finding with continuous time viewed through satisfiability modulo theories (smt). arXiv preprint arXiv:1903.09820 (2019)
23. Turner, J.: Distributed task allocation optimisation techniques. In: Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems. pp. 1786–1787. International Foundation for Autonomous Agents and Multiagent Systems (2018)
24. Zhang, C., Shah, J.A.: Co-optimizating multi-agent placement with task assignment and scheduling. In: IJCAI. pp. 3308–3314 (2016)