Proceedings of the Twelfth International
Symposium on Combinatorial Search (SoCS 2019)

Generalized Target Assignment and Path Finding Using Answer Set Programming

Van Nguyen
Computer Science Department
New Mexico State University

Philipp Obermeier
Computer Science Department
University of Potsdam

Tran Cao Son
Computer Science Department
New Mexico State University

Torsten Schaub
Computer Science Department
University of Potsdam

William Yeoh
Computer Science Department
Washington University in St. Louis

Introduction

Multi-Agent Path Finding (MAPF) deals with teams of agents that need to find collision-free paths from their respective starting locations to their respective goal locations on a graph. This model can be applied to a number of applications (e.g., autonomous warehouse systems (Wurman, D'Andrea, and Mountz 2008)). For example, in an autonomous warehouse system (illustrated by Figure 1), robots (in orange) navigate around a warehouse to pick up inventory pods from their storage locations (in green) and drop them off at designated inventory stations (in purple) in the warehouse. Several extensions of MAPF have been proposed (e.g., *combined Target Assignment and Path Finding* or TAPF).

While TAPF better reflects real-world systems with homogeneous agents, such as our motivating application, it still has a key limitation: It assumes that *the number of agents equals the number of tasks* to be allocated. In our motivating application, there are typically more tasks than agents. As such, agents have to move towards a new task after completing their current task. Therefore, we propose *Generalized TAPF* (G-TAPF), a generalization of TAPF that allows the number of tasks to be *greater* than the number of agents. We also propose a new objective, which better captures more applications including our motivating warehouse application: Each task has an associated deadline that indicates the time at which it must be completed. We also propose use *answer set programming* (ASP) (Lifschitz 2002) as the general framework for solving the new G-TAPF problems.

Generalized TAPF Problems

A *Generalized TAPF* (G-TAPF) problem is given by a triple $P = (G, R, T)$, where

- $G = (V, E)$ is an undirected connected graph, where V and E correspond to locations and ways of moving between locations for the agents;
- R is a set of agents. Each $r \in R$ is specified by a pair (t, s), t is the type of task that can be accomplished by r and $s \in V$ is the starting location of r;
- T is a set of groups of tasks. Each group in T is specified by a set of orders O and a positive integer d representing the deadline of the orders in the group; each $o \in O$ is a pair (g, t) where g and t are the destination and type of the order, respectively.

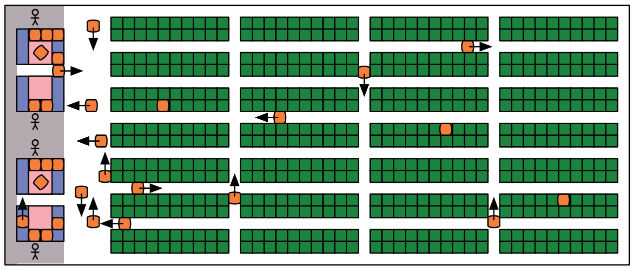
Figure 1: Layout of an Autonomous Warehouse System [Wurman *et al.*, 2008]

For an agent r, $type(r)$ and $loc(r)$ denote the type and starting location of agent r, respectively. For a task t, $type(t)$ and $destination(t)$ denote the type and destination of task t, respectively. For a group T of tasks, $deadline(T)$ denotes the deadline of tasks in group T.

Agents can move between the vertices along the edges of G, one edge at a time, under the restrictions: (*a*) two agents cannot swap locations in a single timestep; and (*b*) each location can be occupied by at most one agent at any time. A path for an agent r is a sequence of vertices $\alpha = \langle v_1, \ldots, v_n \rangle$ if (*i*) the agent starts at v_1 (i.e., $v_1 = loc(r)$); and (*ii*) if for any two subsequent vertices v_i and v_{i+1}, there is an edge between them (i.e., $(v_i, v_{i+1}) \in E$) or they are the same vertex (i.e., $v_i = v_{i+1}$). n is called the length of α and is denoted by $length(\alpha)$.

An agent r completes a task t via a path $\alpha = \langle v_1, \ldots, v_n \rangle$ if $type(r) = type(t)$ and $destination(t)$ is one of the vertices in α, i.e., $destination(t) \in \{v_1, \ldots, v_n\}$. A task is said to be completed when an agent completes it. A group of tasks is completed when every task in the group is completed. A G-TAPF problem P is completed when every group of tasks in T is completed. A *solution* of a G-TAPF problem P is a collection of paths $S = \{\alpha_r \mid r \in R\}$ for the agents in R so that all tasks in T are completed.

Depending on the application, one can create different G-TAPF variants. We describe several variants below:

- *Equal numbers of tasks and agents*: This variant is the original TAPF (Ma and Koenig 2016), where there is a one-to-one allocation of tasks to agents.

- *Group completion*: Agents must complete all groups of tasks in some order. More precisely, for every pair of distinct groups of tasks, all tasks in one group must be completed before all tasks in the other group.
- *Task deadlines*: Agents must complete all the tasks $t \in T_i$ within a group T_i within the deadline of the group $deadline(T_i)$.
- *Completion with checkpoints*: In order to complete a task t, before and/or after reaching the goal $destination(t)$, an agent must visit some other designated checkpoints. Under this view, the autonomous warehouse system (Wurman, D'Andrea, and Mountz 2008) can also be viewed as a G-TAPF variant, where to complete a task t, an agent needs to pick up a pod at a checkpoint before bringing it to the inventory station (= $destination(t)$) and then returning the pod to another checkpoint.

One can optimize different possible objectives:

- The *makespan* of a solution S is defined by $\max_{\alpha \in S} length(\alpha)$. Minimizing this value is appropriate if one wants to minimize the total time taken by the agents to complete all the tasks in the problem. Alternatively, one can also seek to find a solution whose makespan is within a maximum makespan threshold. which is appropriate in problems where there is a deadline in which to complete all the tasks.
- The *total path cost* of a solution S is defined by $\sum_{\alpha \in S} length(\alpha)$. Minimizing this value is appropriate if the cost of a path is measured by fuel consumption and one wants to minimize the total amount of fuel used. As above, one can also seek to find a solution whose total path cost is within a maximum threshold.

Modeling G-TAPFs Using ASP

Let $P = (G, R, T)$ be a G-TAPF problem and n be an integer denoting the upper bound on the solution makespan. Each program P will be represented as an ASP program consisting of different groups of rules:

- *G-TAPF input representation*: This group of rules encodes the given problem P such as the graph G, the groups of tasks and the tasks of P;
- *Task allocation*: This group of rules ensures that each task in P is assigned to exactly one agent of the correct type;
- *Planning rules*: This group of rules is similar to the set of rules for planning developed by the ASP community. It includes (*i*) rules for reasoning about effects of actions (e.g., the locations of the robots will change if they move to the neighbor nodes); (*ii*) inertial rules, i.e., a fluent value does not change if no action that changes the fluent value occurs; (*iii*) rules for generating the action occurrences; (*iv*) rules for preventing actions, that cannot be executed, to occur (e.g., two robots cannot switch locations);
- *Group completion*: This group of rules is introduced when group completion needs to be enforced. It creates an order among the groups of tasks and enforces the completion of the tasks in accordance with the created order.
- *Solution verification, checkpoints, and deadlines*: Rules for solution verification need to check for the achievement of tasks as well as the satisfaction of various requirements such as group completion, deadlines, and checkpoints. When the problem has multiple checkpoints, the rules will ensure that the checkpoints are visited in the order they are specified.

Solving G-TAPFs. The ASP code can be used to compute solutions with makespan n, assuming it exists; or to find a solution with the minimal makespan. Using built-in optimization features of ASP solvers, one can generalize our methods for other objectives (e.g., minimizing the total path cost). Computing optimal solution is computational intensive due to the fact that the number of possible assignments between tasks and robots is exponential in the size of the problem. For this reason, we also propose a greedy strategy for solving G-TAPFs.

Our experimental results show that CBM is better in simple TAPF problems with few conflicts, but worse in difficult problems with more conflicts. For a more complete description of the model, solving process, and experimental results, please refer to the longer version of this paper that appeared at IJCAI 2017 (Nguyen et al. 2017).

Conclusions

Both MAPF and TAPF models suffer from their limiting assumption that the number of agents and targets are equal. In this paper, we propose the Generalized TAPF (G-TAPF) formulation that allows for (1) unequal number of agents and tasks; (2) tasks to have deadlines by which they must be completed; (3) ordering of groups of tasks to be completed; and (4) tasks that are composed of a sequence of checkpoints that must be visited in a specific order. As different G-TAPF variants may be applicable in different domains, we model them using ASP, which allows one to easily customize the desired variant by choosing appropriate combinations of rules to enforce. We show that ASP technologies can easily exploit domain-specific information to improve its scalability and efficiency. The contributions in this paper thus make a notable jump towards deploying MAPF and TAPF algorithms in practical applications.

References

Lifschitz, V. 2002. Answer set programming and plan generation. *Artificial Intelligence* 138(1–2):39–54.

Ma, H., and Koenig, S. 2016. Optimal target assignment and path finding for teams of agents. In *AAMAS*, 1144–1152.

Nguyen, V.; Obermeier, P.; Son, T. C.; Schaub, T.; and Yeoh, W. 2017. Generalized target assignment and path finding using answer set programming. In *IJCAI*, 1216–1223.

Wurman, P.; D'Andrea, R.; and Mountz, M. 2008. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AIMag* 29(1):9–20.