Reasoning and Learning in Real-World Systems for Long-Term Autonomy (LTA)
Papers from the AAAI 2018 Fall Symposium

From Abstract to Executable Models for Multi-Agent Path Finding on Real Robots

Roman Barták, Jiří Švancara, Věra Škopková, David Nohejl
Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic
bartak@ktiml.mff.cuni.cz

Abstract

Multi-agent path finding (MAPF) deals with the problem of finding a collision-free path for a set of agents (robots). An abstract model with a graph describing the environment and agents moving between the nodes of the graph has been proposed. This model is widely accepted by the MAPF community and majority of MAPF algorithms rely on this model. In this paper we argue that the model may not be appropriate, when the plans are to be executed on real robots. We provide some preliminary empirical evidence that abstract plans deviate from real plans executed on robots and we compare several variants of abstract models. The paper motivates further research on abstraction of problems with respect to applicability of solutions in practice.

Introduction

Abstraction is the process of removing details from a problem representation. It is a critical step in problem solving as without abstraction "intelligent agents would be completely swamped by the real world" (Russell and Norvig 2009). Despite its importance, little attention has been paid to abstraction techniques compared to, for example, solving techniques. In areas, such as planning, the formal abstract model has been proposed and many concrete domain models are used for benchmarking, but the studies how to obtain such models and how the models relate to real world are rare.

In this paper, we look at a specific planning problem called *multi-agent path finding* (MAPF) that deals with finding collision-free paths for a set of agents. We selected this problem for several reasons. First, MAPF has a strong practical applicability in areas such as warehousing and intelligent road junctions. Second, there exists a widely-accepted uniform abstract model of MAPF that uses only a few abstract types of actions that are easily executed on real robots.

Our goal is studying appropriateness of MAPF abstract models from the perspective of executing the obtained plans. We will present the core abstract model used by state-of-the-art solvers together with several extensions closer to reality. The obtained plans will be empirically compared by executing them on real robots called Ozobots (Ozobot & Evollve, Inc. 2018). This is a short version of paper (Barták et al. 2018), which gives full technical details. We focus on motivating this type of research and on discussing futures steps.

Background on MAPF

Formally, the MAPF problem is defined by a graph $G = (V, E)$ and a set of agents $a_1, \dots, a_k$, where each agent a_i is associated with starting location $s_i \in V$ and goal location $g_i \in V$. The time is discrete and in every time step each agent can either move from its location to a neighboring location or wait in its current location. A grid map with a unit length of each edge is often used to represent the environment (Ryan 2008). The task is to find a collision-free path for each agent, where the collision occurs when two agents are at the same node at the same time or two agents move along the same edge at the same time in opposite directions. The makespan (the maximal time when all agents reached their destinations) objective function is often studied in the literature (Surynek 2014). The problem to find a makespan-optimal solution is NP-hard (Yu and LaValle 2013). Though the plans obtained by different MAPF solvers might be different, the optimal plans are frequently similar and tight (no superfluous steps are used). Hence, any optimal MAPF solver can be used. We used the reduction-based solver in the Picat programming language (Barták et al. 2017).

MAPF Models and Executable Plans

For our study we designed an environment that is intentionally close to the abstract model of MAPF, that is, it is a grid map with equal distances between vertices that are connected by lines used by robots to easily navigate between the vertices, see Figure 1. The abstract plan outputted by MAPF solvers is, as defined, a sequence of locations that the agents visit. However, a physical agent has to translate these locations to a series of actions that the agent can perform. We assume that the agent can turn left and right and move forward. By concatenating these actions, the agent can perform all the required steps from the abstract plan (recall, that we are working with grid worlds). This translates to five possible actions at each time step - (1) wait, (2) move forward, (3,4) turn left/right and move, and (5) turn back and move. As the mobile robot cannot move backward directly, turning back is implemented as two turns right (or left). Ozobot robots, used in our study, can directly perform these actions, which together with the specific map simplifies typical "robotics" problems such as localization and control.

As the abstract steps may have durations different from the physical steps, the abstract plans, which are perfectly

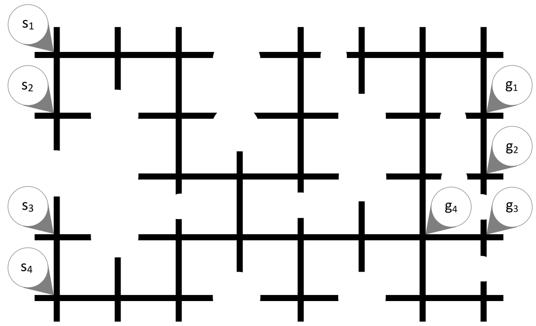

Figure 1: Instance map for Ozobots. Ozobots follow the black line, the gray circles indicate starting and goal locations (not printed on real map).

synchronized, may desynchronize when being executed, which may further lead to collisions. The intuition says that such desynchronization will indeed happen. In our setting, the speed of the robots was set in such a way that moving along a line takes 1600ms and turning takes 800ms. Note that the real robots only blindly follow the computed plan and do not intervene if, for example, an obstacle is detected.

In the rest of the section, we describe the studied abstract MAPF models and possible transformations of abstract plans to executable sequences of physical actions. Let t_t be the time needed by the robot to turn by 90 degrees to either side and t_f be the time to move forward to the neighboring vertex in the grid. Both t_t and t_f are nonzero. The time spend while the agent is performing the wait operation t_w will depend on each model.

Classical Model

The first and most straightforward model is a direct translation of the abstract plan to the action sequence. We shall call this a *classic* model. At the end of each timestep, an agent is facing in a direction. Based on the next location, the agent picks one of the five actions described above and performs it. This means that all move actions consist of possible turning and then going forward. There are no independent turning moves. As the two most common actions in abstract plans are (2) and (3,4), we suggest to set the time t_w of waiting actions to be $t_f + 1/2 * t_t$ as the average of durations of actions (2) and (3,4).

One can easily see that this simple model can be prone to desynchronization, as turning adds time over agents that just move forward. To fix this synchronization issue, we introduce a *classic+wait* model. The basic idea is that each abstract action takes the same time, which is realized by adding some wait time to "fast" actions. The longest action is (5), therefore each action now takes $2 * t_t + t_f$ including the waiting action. The consequence is that plan execution takes longer time, which may not be desirable.

Note that both of these models do not require the MAPF algorithm and model to change. They only use different durations of abstract actions which are implemented in the translation of abstract plans to executable actions.

Robust Model

Another way to fix the synchronization problem is to create a plan π that is robust to possible delays during execution. The k-robust plan is a valid MAPF plan that in addition requires for each vertex of the graph to be unoccupied for at least k time steps before another agent can enter it (Atzmon et al. 2017). In our experiments, we choose k to be 1. We presume that this is a good balance between keeping the agents from colliding with each other while not prolonging the plan too much. The 1-robust plan is then translated to executable actions using the same principle as the *classic* model. This yields a *1-robust* model. Though, this model does not solve the synchronization issue directly, it adds some slack that can prevent collisions caused by various reasons.

Split Actions Model

By making the model less abstract, we can directly represent the executable actions, in particular, by introducing an abstract turning action. In the reduction-based solvers, this can be done by splitting each vertex v_i from the original graph G into four new vertices $v_i^{up}, v_i^{right}, v_i^{down}, v_i^{left}$ indicating directions where the agent is facing to. The new edges now represent the turn actions, while the original edges correspond to move only actions. This change needs to be accompanied by constraints restricting the agents not to be at split vertices at the same time. The abstract plan is then translated to an executable plan in a direct way as the agent is given a sequence of individual actions wait, turn left/right, and move forward. The waiting time t_w is set as the bigger time of the remaining actions: $t_w = \max(t_t, t_f)$. We shall call this a *split* model.

To make the model even closer to reality, we can exploit the weighted MAPF (Barták, Švancara, and Vlk 2018), where each edge in the graph is assigned an integer value that denotes its length. The weighted MAPF solver finds a plan that takes these lengths into account. The lengths of turning edges are assigned a length of t_t and the other edges are assigned a length of t_f (or its scaled value to integers). The waiting time t_w is set as the smaller time of the remaining actions: $t_w = \min(t_t, t_f)$. We shall call this a *weighted-split* model or *w-split* for short.

A final enhancement to the *weighted-split* model is to introduce k-robustness there. This will again ensure that the agents do not tend to move close to each other to avoid undesirable collisions. In this case, however, it is not enough to use 1-robustness, as the plan is split into more time steps. Instead, we use $\max(t_t, t_f)$-robustness. We shall call this *robust-weighted-split* model or *rw-split* for short.

Results of Experiments

We generated plans using each MAPF model for the problem instance described above and then we executed the plans five times in total for each model. Several properties were measured with results shown in Table 1.

Computed makespan is the makespan of the plan returned by the MAPF solver. It is measured by the (weighted) number of abstract actions and this is the value optimized by the solvers. Note that the *split* models have larger makespan

	Comp. Mksp	Failed Runs	#Colls.	Total Time [s]	Max Δ [s]
classic	17	5	4	NA	5
classic+wait	17	0	4.2	53	0
1-robust	19	0	0	41	4
split	27	0	2	36	3
w-split	45	0	2.6	39	0
rw-split	47	0	0	39	0

Table 1: Real performance of Ozobots for studied models.

than the rest because the *split* models use a finer resolution of actions, namely turning actions are included in the makespan calculation. This is even more noticeable with *w-split* and *rw-split*, where the moving-forward action has a duration (weight) of two. Total time is the actual time needed to complete the plan by all robots. To measure the level of desynchronization, we introduced the Max Δ time. We made abstract plans for all robots equally long by adding void wait actions to the end (where necessary). The Max Δ time is the time difference between the real end times of the first and last robots. This value is zero, if the robots remained synchronized during plan execution. The larger value means larger desynchronization. All of the times are rounded to seconds because the measurements were conducted by hand.

The number of failed runs is also shown. The only model that did not finish any run is the *classic* model while the rest managed to finish all of the runs. A run fails if there is a collision that throws any of the robots off the track so the plan cannot be finished. The average number of collisions per run shows how many collisions that did not ruin the plan occurred. These collisions can range from small one, where the robots only touched each other and did not affect the execution of the plan, to big collisions, where the agent was slightly delayed in their individual plan, but still managed to finish the plan. For the *classic* model, where no execution finished, we present the number of collisions occurring before the major collision that stopped the plan.

Conclusions and Future Steps

The goal of the paper is showing that abstract models should be treated more carefully, when the results are supposed to by used in real environment. Our preliminary experiment showed that the most widely used MAPF model, the *classic* one, is actually not applicable even if the environment is made very close to the model. The reason is that durations of real actions are different from durations of abstract actions, which leads to desynchronization of agents' plans. A naive extension to make all actions equally long worsens the quality of plan (makespan) significantly. Adding robustness to abstract plans helps, but as the Max Δ time shows, there is some desynchronization, which may lead to collisions for longer plans. The *split* model uses abstraction closer to reality and adding weights makes the abstract plans even closer to real plans when executed. However, solving such models is more computationally expensive than solving the classical model (Barták, Švancara, and Vlk 2018).

The results show that there is indeed a gab between widely-used theoretical frameworks for MAPF and deployment of solutions in real environments. A wider experimental study is necessary to understand better the relations between abstract models and real environments. For example, the ratio between the length of edges and the size of robots seems important (Ozobots have diameter of 3 cm and distance between nodes in our map is 5 cm). Note also, that blind execution of plans was assumed. It would be interesting to look at plan-execution policies that assume communication between agents and exploit information from sensors (Ma, Kumar, and Koenig 2017).

Acknowledgements

Roman Barták is supported by the Czech Science Foundation under the project P202/12/G061 and together with Jiří Švancara by the Czech-Israeli Cooperative Scientific Research Project 8G15027.

References

Atzmon, D.; Felner, A.; Stern, R.; Wagner, G.; Barták, R.; and Zhou, N. 2017. k-robust multi-agent path finding. In *Proceedings of the Tenth International Symposium on Combinatorial Search (SoCS)*, 157–158.

Barták, R.; Zhou, N.-F.; Stern, R.; Boyarski, E.; and Surynek, P. 2017. Modeling and solving the multi-agent pathfinding problem in picat. In *29th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, 959–966. IEEE Computer Society.

Barták, R.; Švancara, J.; Škopková, V.; and Nohejl, D. 2018. Multi-agent path finding on real robots: First experience with ozobots. In *Advances in Artificial Intelligence – IBERAMIA 2018*. Springer.

Barták, R.; Švancara, J.; and Vlk, M. 2018. A scheduling-based approach to multi-agent path finding with weighted and capacitated arcs. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, 748–756.

Ma, H.; Kumar, T. K. S.; and Koenig, S. 2017. Multi-agent path finding with delay probabilities. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, 3605–3612. AAAI Press.

Ozobot & Evollve, Inc. 2018. Ozobot — Robots to code, create, and connect with.

Russell, S., and Norvig, P. 2009. *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Ryan, M. R. K. 2008. Exploiting subgraph structure in multi-robot path planning. *J. Artif. Intell. Res.* 31:497–542.

Surynek, P. 2014. Compact representations of cooperative path-finding as SAT based on matchings in bipartite graphs. In *26th IEEE International Conference on Tools with Artificial Intelligence, ICTAI*, 875–882. IEEE Computer Society.

Yu, J., and LaValle, S. M. 2013. Structure and intractability of optimal multi-robot path planning on graphs. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*.