Situated Grid Pathfinding Among Moving Obstacles (Extended Abstract)

Devin Thomas,[1] **Tianyi Gu,**[1] **Wheeler Ruml,**[1] **Eyal Solomon Shimony**[2]

[1]Department of Computer Science, University of New Hampshire, USA
[2]Department of Computer Science, Ben-Gurion University of the Negev, Israel
devin.thomas@unh.edu, gu@cs.unh.edu, ruml@cs.unh.edu, shimony@bgu.ac.il

Introduction

We present a preliminary study of Situated Pathfinding Among Moving Obstacles (SPAM-O), in which the objective is to traverse a grid, reaching the goal as quickly as possible while avoiding both static and moving obstacles. Real-time search planning algorithms address this by setting a fixed time bound for the agent to return an incremental plan. In contrast, in the situated planning problem setting, the agent plans "as the clock ticks" (Cashmore et al. 2018), meaning that time passes whether the agent is moving, thinking, or waiting. In situated planning, the agent can plan and execute concurrently, and thus may benefit from committing to a longer duration action, as that allows more time to plan (Cserna, Ruml, and Frank 2017). A similar example of a situated problem setting is the video game Frogger, in which a vulnerable frog tries to cross a busy roadway.

To address SPAM-O, we examine ideas from real-time planning and safe interval path planning (SIPP) (Phillips and Likhachev 2011) and explore which combinations of methods are most successful for a situated agent. We develop and test a new 'subinterval'-based method for successor generation and state heuristic generalization. We find that subinterval-based approaches increase the success rate of situated agents with low to moderate expansion rates, and that partitioned learning is critical for a successful agent.

SPAM-O

The SPAM-O state space is $\langle x, y, t\rangle$ where the location is discrete and time is continuous. The environment contains static and moving obstacles, which are represented by safe temporal intervals at each grid cell. The actions are of the form *wait and move*: waiting for a specified real-valued amount of time and then optionally moving to one of the 8 adjacent grid cells. The agent seeks to minimize goal achievement time: the total time from when the problem is presented until the agent arrives at the goal. Actions cost their duration and when the agent moves it is with a constant speed. A SPAM-O problem is a 6-tuple: $\langle$states $\in S$, safe intervals $I(x, y)$, actions $\in A$, expansion rate E, $s_{start} \in S$, goal location $G\rangle$. The expansion rate E is the number of expansions done by the search algorithm per time unit. The agent has perfect information, but its time to look at this information is limited by the expansion rate E.

SPAM-O is a situated version of the same problem addressed by SIPP. Safe intervals are constructed by grouping all consecutive co-located states into a safe interval at that location, $i \in I$. This compresses the continuous time dimension into a compact discrete representation, allowing optimal solution with A^* (Phillips and Likhachev 2011), or sub-optimally with weighted A^* or focal search variants (Yakovlev, Andreychuk, and Stern 2020), or anytime algorithms sufficiently fast to be used in soft-real-time on a specific problem instance (Narayanan, Phillips, and Likhachev 2012).

SIPP and its variants are able to search over a search space $\langle x, y, i\rangle$ of safe intervals because, when searching offline, earlier arrival within a safe interval is always at least as good as any later arrival, as the agent could wait in place to reach that later state. This is no longer the case when the agent is situated, as a hasty agent that forgoes extra planning by moving as quickly as possible into a safe interval may miss out on opportunities that require more deliberation to recognize. SPAM-O requires a more sophisticated handling of intervals to be correct.

We also examine how leading real-time search methods fare in the similar setting of situated planning. The situated agent must perform heuristic learning in order to escape local minima; one can back up information from the search frontier using methods like local search space real-time A*(LSS-LRTA*) (Koenig and Sun 2009), potentially with separate learning of costs due to static versus dynamic portions of the environment using partitioned learning real-time A*(PLRTA*) (Cannon, Rose, and Ruml 2014).

Algorithms

We explore three facets of SPAM-O: 1) the successor generation strategy, which controls which finite subset of the potentially infinite possible successor states are generated; 2) the heuristic generalization strategy, which controls how heuristic learning is generalized over related states; and 3) the heuristic learning strategy, which controls how the agent propagates heuristic information from the search frontier. In offline SIPP, facets 2 and 3 do not apply, and the first is avoided, due to earlier states within a safe interval domi-

nating, allowing search to treat safe intervals as atomic. We term our problem SPAM-O instead of Situated SIPP to highlight that SIPP is not the only state space representation one may use.

The first successor generation strategy, 'Earliest in Interval' (EI-n) generates actions by selecting the successor with the minimum wait, and thus arrives as early in the destination safe interval as possible. The next n safe intervals of each adjacent cell are considered, thus EI-∞ is equivalent to SIPP. The second, 'Punctual Interval' (PI-n) generates actions with as long a wait as possible, while still passing through the same safe intervals, on the way to the same frontier state as EI-n. PI-n maximizes the time that the agent has to plan on the way to its child. Both EI-n and PI-n generate actions consistent with an optimal plan.

A generalization method prescribes which states share heuristic information. NoGen does no generalization, the 2d-grid of our setting makes it more likely for the agent to find many paths through the same state. This makes learning without generalizing more viable than it would be in a setting where states are unlikely to be re-encountered. Interval Generalization (IGen) stores a single scalar value that is shared by all the states within each safe interval; this is inadmissible. Subinterval Generalization (SubGen) represents heuristic values as a piecewise-linear function of the departure time from a state. As explained more below, each piece represents moving to a particular successor during a particular safe 'subinterval' $\langle begin, end, h \rangle$:

$$h(t) = \begin{cases} \infty & t > end \\ h & begin \leq t \leq \text{end} \\ h + begin - t & t < begin \end{cases} \quad (1)$$

Our first learning method, NoLearn, does no learning. Our second is an adaptation of the learning done by LSS-LRTA*. The third method, PLRTA*, augments LSS-LRTA* by partitioning the h value into a portion due to static obstacles (and hence generalizable) and dynamic obstacles (unique to an interval). Any time spent waiting is due to dynamic obstacles, we blithely assume the rest is static. This may result in learning inadmissible static heuristic values when the search is forced to avoid a dynamic obstacle; defining an admissible partitioning scheme for SPAM-O is on-going work.

The information required for PI-n and SubGen can be tracked using subintervals. For a subinterval $\langle begin, end, h \rangle$, begin corresponds to the earliest time the agent can follow the partial plan without additional waiting, and end corresponds with the deadline to depart while still following the partial plan. To track subintervals we propagate them from the search frontier up through the search tree. A frontier safe interval has a single subinterval extending from the beginning to the end of the safe interval, with h the conventional admissible cost-to-go. As we move up the search tree, the set of subintervals of the parent is the union of the subintervals of each of its children, shifted forward in time and up in heuristic cost-to-go by the duration of the action to move from the parent to the child. The set of subintervals of each successor interval gives us the information we need for PI-n, and carries the heuristic values we need for SubGen.

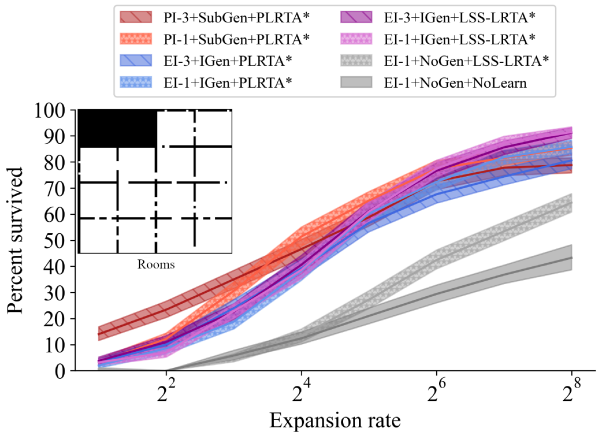

Figure 1: Experimental results, the solid lines are measured success rate with the area between the 5th and 95th percentile shaded, determined by bootstrapping with n = 1024.

Experiments

To evaluate these methods, we used problem instances generated by Yakovlev, Andreychuk, and Stern (2020). 500 instances with 150 moving obstacles each were generated on the rooms static map (see Figure 1 inset). Each generated moving obstacle's trajectory avoids prior obstacles. We generate safe intervals from these instances by calculating when the circular agent is safe from all the circular obstacles at each grid point. We avoid collision checking in generating safe intervals (for an agent moving in a continuous manner), where temporal extents of a safe interval depends on the direction of movement of the agent. Similar to a Star Trek transporter beaming our agent from the source to the destination, we require that both locations be safe for the duration of the action. Tested configurations were: for successor generation: PI-1, PI-3, EI-1, EI-3; for generalization: PI-n + SubGen, EI-n + IGen, EI-n + NoGen and for learning: PLRTA*, LSS-LRTA* and NoLEarn.

In Figure 1, we see a low success rate until expansion rates are high with even the most successful methods, meaning that our benchmark is challenging for a situated agent. PLRTA* and LSS-LRTA* + IGen can learn the static environment and outperform methods that do not, such as NoLearn or LSS-LRTA* + NoGen. This implies that generalizing across states is important for situated agents in environments with local minima, such as the rooms map. Finally we see PI-3 + SubGen + PLRTA* survive significantly more than other methods at low expansion rates, and PI-1 + SubGen + PLRTA* agents survive significantly more than those using other successor generation methods at moderate expansion rates. This result suggests that, with low expansion budgets, agents can benefit from using the subinterval based methods.

Acknowledgements

This research was supported by grant 2019730 from the United States-Israel Binational Science Foundation (BSF) and grant 2008594 from the United States National Science Foundation (NSF).

References

Cannon, J.; Rose, K.; and Ruml, W. 2014. Real-time Motion Planning with Dynamic Obstacles. *AI Communications*, 27: 345–362.

Cashmore, M.; Coles, A.; Cserna, B.; Karpas, E.; Magazzeni, D.; and Ruml, W. 2018. Temporal planning while the clock ticks. In *ICAPS-28*.

Cserna, B.; Ruml, W.; and Frank, J. 2017. Planning Time to Think: Metareasoning for On-line Planning with Durative Actions. In *ICAPS-27)*.

Koenig, S.; and Sun, X. 2009. Comparing real-time and incremental heuristic search for real-time situated agents. *Autonomous Agents and Multi-Agent Systems*, 18(3): 313–341.

Narayanan, V.; Phillips, M.; and Likhachev, M. 2012. Anytime Safe Interval Path Planning for dynamic environments. In *IROS-2012*, 4708–4715.

Phillips, M.; and Likhachev, M. 2011. SIPP: Safe interval path planning for dynamic environments. In *ICRA-2011*, 5628–5635. IEEE.

Yakovlev, K.; Andreychuk, A.; and Stern, R. 2020. Revisiting Bounded-Suboptimal Safe Interval Path Planning. In *ICAPS-30*, 300–304. AAAI Press.