X*: Anytime Multiagent Path Planning With Bounded Search

Extended Abstract

Kyle Vedder
College of Information and Computer Sciences
Amherst, Massachusetts
kvedder@umass.edu

Joydeep Biswas
College of Information and Computer Sciences
Amherst, Massachusetts
joydeepb@cs.umass.edu

ABSTRACT

Multi-agent planning in dynamic domains is a challenging problem: the size of the configuration space increases exponentially in the number of agents, and plans need to be re-evaluated periodically to account for moving obstacles. However, we have two key insights that hold in several domains: 1) conflicts between multi-agent plans often have *geometrically local* resolutions within a small repair window, even if such local resolutions are not globally optimal; and 2) the partial search tree for such local resolutions can then be iteratively improved over successively larger windows to eventually compute the global optimal plan. Building upon these two insights, we introduce 1) a class of anytime multiagent planning solvers, 2) a naïve solver in this class, and 3) an efficient solver in this class which reuses prior search information when improving a solution.

KEYWORDS

multiagent planning; anytime planning; bounded search; search reuse; anytime multiagent planning

ACM Reference Format:
Kyle Vedder and Joydeep Biswas. 2019. X*: Anytime Multiagent Path Planning With Bounded Search. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019*, IFAAMAS, 3 pages.

1 INTRODUCTION AND RELATED WORK

Quickly constructing collision-free paths from a start to a goal is a problem faced by almost all robotic systems in dynamic environments. Adding more agents makes this problem exponentially harder [3], causing this problem, known as the Multiagent Planning Problem (MPP), to be pressing for many multiagent systems. Various planners exist to solve the MPP[1, 2, 4, 5, 7, 9]; however, in this work we are interested in planners which produce optimal solutions, in particular M* and CBS.

M* [8] is a state-of-the-art MPP solver that computes an optimal policy for each agent in individual space, constructs a path in joint space from the individual policies, and then uses the individual policies to inform local repairs to the joint space path when interactions are detected. In sparse domains, the dimensionality of these repairs are low, allowing M* to quickly solve the MPP.

This work is supported in part by AFRL and DARPA under agreement #FA8750-16-2-0042, and NSF grant IIS-1724101.
Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 13–17, 2019, Montreal, Canada.

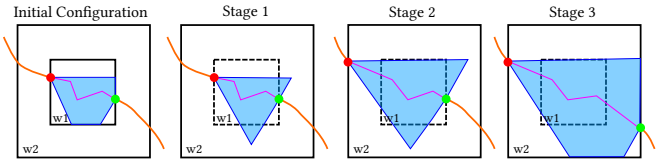

Figure 1: The three stages of X*'s grow and replan algorithm which allow it to save computation between searches. The red dot is the start, the green dot is the goal, the blue area is the search tree, the smaller box is the old window, the larger box is the new window, the purple path is the search solution, and the orange lines are non-colliding joint space paths. Initial Configuration to Stage 1 removes the restriction of the smaller window in the search from the old start to old smaller goal, Stage 1 to Stage 2 moves the start from the old start to the new start, and Stage 2 to Stage 3 moves the goal from the old goal to the new goal.

Another state-of-the-art planner, Conflict Based Search (CBS) [6] approaches the MPP differently. CBS builds a conflict graph, modeling different worlds with constraints, and replans with these constraints in each agent's individual space. This approach allows for planning space to grow exponentially in the number of conflicts rather than the number of agents. In sparse domains, the number of these conflicts is low, allowing CBS to quickly solve the MPP.

2 CONTRIBUTIONS

In this work we present 1) SWP, a class of anytime MPP solvers, 2) Naïve Window A* (NWA*), a naïve SWP solver, and 3) Expanding A* (X*), an efficient SWP solver.

2.1 Simple Windowed Planner

SWP is a class of anytime MPP solvers that leverage search bounding for fast, anytime plan generation. Shown in Algorithm 1, SWP solvers operate by first planning for each agent independently, and then identifying interacting groups of agents from these individual plans. Next, for each interacting group, SWP solvers project the individual plans into the joint planning space of the group, and construct a joint space *window*, an artificial geometric bound, around the point of interaction. SWP solvers then proceed to repair the collision in this window. While the time budget is not exhausted, SWP solvers then iteratively grow the windows and replan inside them, thereby improving the quality of the existing plan.

2.2 Naïve Window A*

Naïve Window A* (NWA*) is a naïve SWP solver. It defines a window to be a set of contiguous states in joint agent space. It possesses a set of interacting agents and a start b and goal e in the joint space

Algorithm 1 Simple Windowed Planner

1: **procedure** SWP
2: $\Pi \leftarrow$ independently planned paths for all agents
3: $S \leftarrow$ interacting set(s) of agents in Π
4: $W \leftarrow \{w \mid w$ is the smallest fit window for $s \in S\}$
5: $\forall w \in W$, plan jointly in w, update Π
6: **while** more time available **do**
7: $\forall w \in W$, grow w and replan jointly, update Π

of these agents agents. In addition, for an arbitrary window w_1, it has a successor, w_2, where $w_1 \subset w_2$. For our experimentation, a window is characterized by a center state, and a radius; it contains all states with an L_∞ norm less than or equal to the radius. The successor of a window is implemented by incrementing its radius.

An initial window size is selected for all new windows, and these windows are produced on SWP's line 4. To form a valid solution, NWA* runs A* inside the window in the joint space of the involved agents, and updates a section of the individually planned solution with the replanned segment in the window. To grow the window in SWP's line 7, NWA* replaces each window with its successor and replans from scratch, again updating a section of the individually planned solution with the replanned segment in the window.

2.3 Expanding A*

Like NWA*, Expanding A* (X*) is an SWP solver. It uses the same window definition and initial planning strategy as NWA*; however, for SWP's line 7 it is able to efficiently reuse information from the prior search in the next search, speeding up successive solution generation. To reuse information while growing and replanning in a window, X* employs a three stage solution, shown in Figure 1.

These three stages operate much like standard A*; they use an open list, O, to hold the search frontier, and a closed list, C, to hold already expanded states, with states $s \in O$ expanded in the order of minimum f-value, $f(s)$, with this minimum state accessed by top(O). They also have a state neighbor function, $N(s)$, which returns the set of collision-free neighbors of s. In addition, it also uses the unique concept of an "out-of-window" list, X, which stores states removed from O and intended to be expanded, but are outside of the current window boundary. These states are stored in X for use in the next search. Finally, Stage 3 reasons about the path between the successive window starts b_2 and b_1 along the path, π, and accesses the cost of this path via $\|\pi\|$.

Figure 1 shows the three stages of X*'s grow and replan algorithm (SWP's line 7). Stage1 transforms a search tree from b_1 to e_1 in w_1 into a search tree from b_1 to e_1 in w_2. Stage2 transforms the search tree from b_1 to e_1 in w_2 into a search tree from b_2 to e_1 in w_2. Stage3 transforms the search tree from a from b_2 to e_1 in w_2 into a search tree from b_2 to e_2 in w_2.

3 RESULTS AND CONCLUSION

To demonstrate X*'s performance, we compared it against Operator Decomposition (OD) M*[1] and CBS[2], using the metric of Normalized Runtime, a 95% CI over 100 trials of algorithm runtime divided by the runtime of an individual space A* search for each agent, in order to normalize across implementation quality.

[1]M* Source Code URL: https://github.com/gswagner/mstar_public
[2]CBS Source Code URL: https://github.com/whoenig/libMultiRobotPlanning

Algorithm 2 X* Algorithms

1: **procedure** A*SearchUntil(O, C, X, w, f_{max})
2: **while** $f(\text{top}(O)) < f_{max}$ **do**
3: $s \leftarrow \text{top}(O)$; $O \leftarrow O \setminus \{s\}$
4: **if** $\exists s' \in C : s = s' \wedge f(s) \geq f(s')$ **then continue**
5: **if** $s \notin w$ **then** $X \leftarrow X \cup \{s\}$ **continue**
6: $C \leftarrow C \cup \{s\}$; $O \leftarrow O \cup N(s)$

1: **procedure** Stage1
2: $O \leftarrow O \cup X$; $X \leftarrow \emptyset$
3: A*SearchUntil($O, C, X, w_2, f(e_1)$)

1: **procedure** Stage2
2: **for all** $s \in O, C$ **do** $f(s) \leftarrow f(s) + \|\pi\|$
3: **for all** $s \in \pi$ **do** $C \leftarrow C \cup \{s\}$; $O \leftarrow O \cup N(s)$
4: A*SearchUntil($O, C, X, w_2, f(e_1) + \|\pi\|$)

1: **procedure** Stage3
2: **for all** $s \in O, C$ **do** $h(s) \leftarrow H(s, e_2)$
3: **while** $O \neq \emptyset$ **do**
4: $s \leftarrow \text{top}(O)$
5: **if** $s = e_2$ **then return** UnwindPath(C, e_2, b_2)
6: $O \leftarrow O \setminus \{s\}$
7: **if** $s \in C$ **then continue**
8: **if** $s \notin w$ **then** $X \leftarrow X \cup \{s\}$; **continue**
9: $C \leftarrow C \cup \{s\}$; $O \leftarrow O \cup N(s)$
10: **return** NOPATH

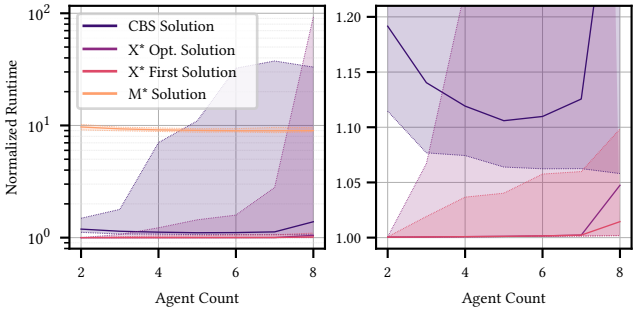

Figure 2: Normalized runtime of X* and state-of-the-art on a 5000mm × 4000mm section of a randomized RoboCup SSL field with stationary opponents; 95% confidence intervals over 100 trials. Left is a full plot, right is an enlarged section of the left plot between 1 and 1.2 of Normalized Runtime.

Figure 2 shows the performance of X* both as an optimal MPP solver and an anytime MPP solver. X* is able to very quickly generate a first solution while generating optimal solutions competitive with the state-of-the-art, and a median optimal runtime below the confidence intervals of M* or CBS. This experimentation suggests that X* is a viable as an optimal MPP solver that also provides anytime properties in domains with sparse interactions, and positions SWP solvers as an exciting new area of MPP research.

REFERENCES

[1] Liron Cohen, Matias Greco, Hang Ma, Carlos Hernández, Ariel Felner, T. K. Satish Kumar, and Sven Koenig. 2018. Anytime Focal Search with Applications. In *IJCAI*.

[2] Ariel Felner, Roni Stern, Solomon Eyal Shimony, Eli Boyarski, Meir Goldenberg, Guni Sharon, Nathan R. Sturtevant, Glenn Wagner, and Pavel Surynek. 2017. Search-Based Optimal Solvers for the Multi-Agent Pathfinding Problem: Summary and Challenges. In *SOCS*.

[3] J.E. Hopcroft, J.T. Schwartz, and M. Sharir. 1984. On the Complexity of Motion Planning for Multiple Independent Objects; PSPACE - Hardness of the "Warehouseman's Problem". In *The International Journal of Robotics Research*. 76–88.

[4] M Renee Jansen and Nathan R. Sturtevant. 2008. Direction Maps for Cooperative Pathfinding. *Proceedings of the 4th Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE 2008*.

[5] Malcolm Ross Kinsella Ryan. 2008. Exploiting Subgraph Structure in Multi-Robot Path Planning. *J. Artif. Intell. Res.* 31 (2008), 497–542.

[6] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219 (2015), 40 – 66.

[7] Guni Sharon, Roni Stern, Meir Goldenberg, and Ariel Felner. 2011. The Increasing Cost Tree Search for Optimal Multi-agent Pathfinding. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume One (IJCAI'11)*. AAAI Press, 662–667.

[8] G. Wagner. 2015. *Subdimensional Expansion: A Framework for Computationally Tractable Multirobot Path Planning*. Ph.D. Dissertation. The Robotics Institute Carnegie Mellon University.

[9] Ko-Hsin Cindy Wang and Adi Botea. 2008. Fast and Memory-efficient Multi-agent Pathfinding. In *Proceedings of the Eighteenth International Conference on International Conference on Automated Planning and Scheduling (ICAPS'08)*. AAAI Press, 380–387.