A Conflict Avoidance Table for Continuous Conflict-Based Search (Extended Abstract)

Vianney Coppé, Pierre Schaus

UCLouvain
{vianney.coppe,pierre.schaus}@uclouvain.be

Abstract

Conflict-Based Search is a state-of-the-art algorithm solving the Multi-Agent Path Finding problem. Given multiple agents with start and goal locations, the problem is to find a set of collision-free paths of minimal cost. Continuous Conflict-Based Search is a recent adaptation of this algorithm for continuous time and agents with physical shapes. However, an important ingredient has not been adapted to this continuous version: the Conflict Avoidance Table. It is used as a tie-breaking strategy in single-agent search phases to favor paths causing fewer conflicts with the other agents. This paper explains how the R-Tree can be used as a Conflict Avoidance Table for Continuous Conflict-Based Search. The experiments show that using the Conflict Avoidance Table can reduce the number of nodes expanded by the algorithm by a large margin. As a result, the solving time is improved proportionally and especially when using the implementation based on R-Trees as opposed to a naive implementation.

Introduction

The Multi-Agent Path Finding problem (MAPF) concerns the planning of collision-free paths on an undirected graph with unit-cost edges $\mathcal{G} = (V, E)$ for n agents with start and goal locations, $S : \{1, \ldots, n\} \to V$ and $G : \{1, \ldots, n\} \to V$ respectively. At each timestep, the agents can either stay at their current location or transition to a neighbor vertex. The goal of MAPF is to minimize the *sum-of-costs* i.e. the sum of all agents' path lengths. While this standard MAPF formulation considers time discretized in unit timesteps and agents as point objects, a more realistic variant is the MAPF with continuous time, denoted MAPF$_R$ (Andreychuk et al. 2022). It allows actions to connect two vertices with arbitrary motion functions with non-unit durations and takes into account the physical shape of the agents.

Conflict-Based Search (CBS) (Sharon et al. 2015) is among the fastest optimal algorithms to solve the MAPF. In (Andreychuk et al. 2022), Continuous CBS (CCBS) generalized this algorithm to solve the MAPF$_R$. However, a feature of CBS that was not migrated to CCBS is the *conflict avoidance table* (CAT). The CAT is used as a tie-breaking strategy in single-agent path finding searches to prefer paths leading to fewer conflicts with the other agents. In CBS, a lookup table indexed by time–position pairs can be used for this purpose. As the MAPF$_R$ lifts some of the simplifying assumptions of MAPF, the CAT needs to be represented by another data structure. After summarizing how CBS and CCBS work, we explain how R-Trees (Guttman 1984) can be used as a CAT for MAPF$_R$. Finally, we present the results of our computational experiments before concluding the paper.

Conflict-Based Search

CBS (Sharon et al. 2015) is a two-level search procedure where the high-level search explores a *constraint tree* (CT) and the low-level search solves single-agent path finding problems given the constraints of a CT node. A *constraint* $\overline{\langle i, a, t\rangle}$ prevents agent i of performing action a at time t. At the start of the algorithm, the root of the CT is created with an empty set of constraints and a plan containing the shortest path of each agent. Then, CT nodes are expanded in best-first order, based on their plan cost. When a CT node is processed, conflicts are detected in the plan it contains. If no conflicts are found, then the plan is an optimal solution. Otherwise, one of the conflicts found is selected according to some criteria. For a conflict $\langle a_i, a_j, t\rangle$ happening at time t between actions a_i and a_j of agents i and j respectively, two child nodes are created. On top of the constraints of the parent CT node, the conflict is resolved by adding the constraint $\overline{\langle i, a_i, t\rangle}$ to the left child and $\overline{\langle j, a_j, t\rangle}$ to the right child. In both child nodes, a new path is computed by the low-level search using space-time A* for the agent with the additional constraint, they are then added to the set of open CT nodes.

CCBS (Andreychuk et al. 2022) is an adaptation of CBS for the MAPF$_R$. The major difference with respect to CBS is that conflict resolution is more complex. Given a conflict $\langle a_i, t_i, a_j, t_j\rangle$ between actions a_i and a_j starting at time t_i and t_j, CCBS starts by computing the *unsafe interval* of each action. The unsafe interval $[t_i, t_i^u)$ of action a_i is the maximal time interval during which starting the action would conflict with a_j performed at time t_j. Then, the conflict is resolved by adding constraints to the child CT nodes, which prevent the agents from starting the actions within the unsafe intervals computed. In the low-level search, Safe Interval Path Planning (Phillips and Likhachev 2011) can be used to find single-agent paths respecting time interval constraints without needing to discretize time.

Conflict Avoidance

After creating a CT node with an additional constraint for one agent, the low-level search is called to find a path respecting all its constraints. At this point, CBS applies the mechanism of *conflict avoidance* introduced in (Standley 2010). The idea is store all actions of the other agents' paths in a CAT used to quickly detect if moves being planned conflict with any of them. A counter is kept in each node of the search algorithm, which is then used as a tie-breaking rule for nodes with the same f-value: nodes with fewer conflicts will be preferred. As a result, the low-level search will favor paths creating fewer conflicts with all other agents. By doing so, some conflicts can be avoided and it prevents useless branching in the high-level search.

In the case of MAPF$_R$, conflicts can occur between actions that do not happen on the same edge or vertex since agents have a physical shape and can follow complex motion functions. In addition, time is not discretized. Consequently, a CAT cannot be a simple lookup table indexed by vertex or edge and time, as for the MAPF.

R-Trees R-Trees were first introduced in (Guttman 1984) to efficiently store and query spatial data. As its name suggests, the R-Tree is a tree data structure representing a hierarchy of n-dimensional axis-aligned *minimum bounding boxes* (MBBs). R-Trees are capable of answering intersection and containment queries, and also of retrieving the k-nearest neighbors of a geometry. They are widely used in database systems including geographic information systems (GIS), multimedia databases and spatiotemporal databases.

Storing and querying actions We represent timed actions as 3-dimensional MBBs with one temporal dimension and two spatial dimensions. On the time axis, the extremities of the MBB are simply the start and end times of the interval during which the action takes place. For the spatial axes, the MBB should enclose the physical shape of the agent positioned at the extreme points of the action's trajectory. Using these MBBs, we can fill the R-Tree with all the existing timed actions in a plan and query it in the low-level search of CCBS with the timed actions an agent wishes to perform. When planning an action in the low-level search, its MBB is created and used to query the R-Tree. However, the MBBs are rough approximations of the trajectories of the agents. Thus, when queries return some timed actions, we must perform the real collision detection procedure, as in two-phase collision detection (LaValle 2006).

Computational Experiments

To evaluate the impact of the CAT on the performance of CCBS, we extended the source code developed for the paper (Andreychuk et al. 2022) with two versions of CAT. The naive implementation simply loops through all actions of the plan and performs collision detection for actions overlapping in time. The other uses the R-Tree implementation from the Boost libraries (Boost 2022). We replicated the experimental process of (Andreychuk et al. 2021) with maps and scenarios taken from the Moving AI repository (Sturtevant 2012). For each map, experiments are made with 2^k-neighborhoods with k ranging from 2 to 5. The algorithms have 5 minutes to solve each scenario.

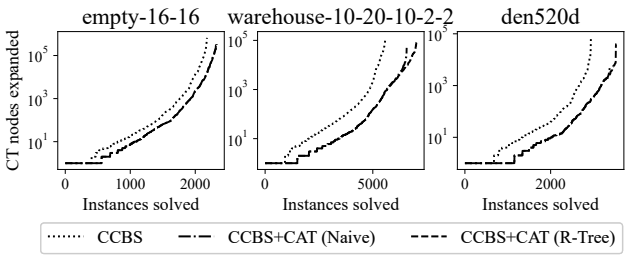

Figure 1: Number of instances solved by each algorithm within a given number of CT nodes expanded.

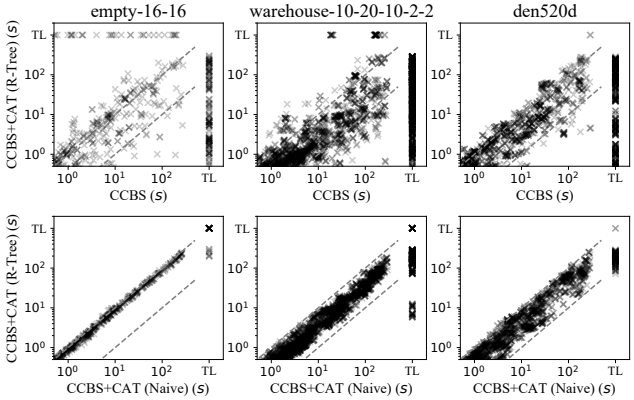

Figure 2: Comparison of the runtime for each instance.

Figure 1 compares the number of instances solved by each algorithm within a fixed number of CT nodes expanded. Except for some instances not solved by the naive implementation, the lines for CCBS+CAT (Naive) and CCBS+CAT (R-Tree) coincide as they both implement the same conflict avoidance behavior. Compared to CCBS, the number of CT nodes expanded by either version of CCBS+CAT is reduced significantly. The first row of Figure 2 compares the runtime of CCBS+CAT (R-Tree) and CCBS for each individual instance. CCBS+CAT (R-Tree) is able to solve many more instances than CCBS and is generally faster, especially on larger maps – the first map is 16×16 while the two others are 161×63 and 256×257. There is little difference between the two versions of CCBS+CAT for the first map probably because the paths are short. For larger maps, however, using R-Trees is almost always faster and reaches speedups of one order of magnitude in some cases.

Conclusion

This paper showed how R-Trees can be used to implement an efficient CAT allowing for actions of any duration and rules of motion, and agents with arbitrary physical shapes. The experiments demonstrated the positive impact of adding a CAT to CCBS: reducing the number of CT nodes and improving the overall computation time of the algorithm.

Acknowledgments

We would like to thank the authors of (Andreychuk et al. 2022) for making their source code publicly available.

References

Andreychuk, A.; Yakovlev, K.; Boyarski, E.; and Stern, R. 2021. Improving Continuous-time Conflict Based Search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(13): 11220–11227.

Andreychuk, A.; Yakovlev, K.; Surynek, P.; Atzmon, D.; and Stern, R. 2022. Multi-agent pathfinding with continuous time. *Artificial Intelligence*, 305: 103662.

Boost. 2022. Boost C++ Libraries. boost.org. Accessed: 2022-06-01.

Guttman, A. 1984. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, 47–57.

LaValle, S. M. 2006. *Planning algorithms*. Cambridge university press.

Phillips, M.; and Likhachev, M. 2011. Sipp: Safe interval path planning for dynamic environments. In *2011 IEEE International Conference on Robotics and Automation*, 5628–5635. IEEE.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219: 40–66.

Standley, T. 2010. Finding Optimal Solutions to Cooperative Pathfinding Problems. *Proceedings of the AAAI Conference on Artificial Intelligence*, 24(1): 173–178.

Sturtevant, N. R. 2012. Benchmarks for grid-based pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2): 144–148.