EECBS for Nonholonomic Agents

Akshaya KS Harshit Agrawal Manuj Trehan Vrushali Patil Nevin Valsaraj

Abstract

Multi-Agent Path Finding (MAPF) is an important problem where collision-free paths for multiple robots are to be computed in a small amount of time. Conflict-based search (CBS) **[1]** is a two-level algorithm that solves the MAPF problem optimally. In CBS, the path for each agent is computed independently, and then the collisions between two agents are resolved by branching. Each branch is a new candidate plan wherein one agent or the other is forced to find a new path that avoids the given collision. Enhanced CBS (ECBS) **[2]** is an extension of CBS that uses focal search and speeds up the original CBS algorithm. However, the solution obtained is epsilon times sub-optimal. Explicit Estimation CBS (EECBS) **[3]** is another variant of CBS that decreases the runtime even further, but it is a bounded suboptimal solution. It uses online learning to obtain inadmissible estimates of the cost of the solution of each high-level node and uses EES **[4]** to choose which high-level node to expand next. The current algorithm and implementation of EECBS operate on (x,y) state space. This project aims to extend this implementation to include θ in the state space. This was done keeping in mind that most robots have non-holonomic constraints and a lattice-based planner would be more appropriate for such use cases.

Introduction

The multi-agent path-finding (MAPF) problem has been studied thoroughly in the literature. Though CBS is a leading algorithm that solves MAPF optimally, it is slow and computationally expensive. MAPF in general, is an NP-hard problem to solve optimally **[5]**; thus, this has motivated the research community to find bounded-suboptimal algorithms that reduce the runtimes of the search. EECBS is one such variant of CBS, and we are extending it for nonholonomic agents by making the state space as (x, y, θ).

There are generally two levels in this search algorithm. The top level is a constraint tree that addresses the collisions between agents starting from the optimal path of each agent. The second level is then a low-level planner that replans the path for each agent adhering to the constraints. Fig. 1 below shows the optimal Space-Time-Astar path for each agent without conflict resolution. CBS then tries to address each conflict that arises till there are no more conflicts. EECBS uses Explicit estimation search to solve it sub-optimally but with significant speed-up. Our primary motivation was to evaluate the algorithm on more realistic robots, which is covered in-depth in the methodology section.

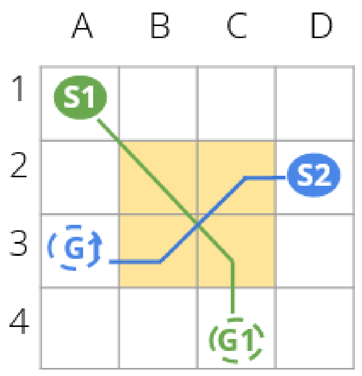


Fig. 1: MAPF in (x, y, θ) state space

Method

Explicit Estimation CBS (EECBS) is a new bounded-suboptimal variant of CBS that uses online learning to obtain inadmissible estimates of the cost of the solution of each high-level node and uses EES to choose which high-level node to expand next. Motivated by Explicit Estimation Search, EECBS decreases its runtime even further using inadmissible heuristics. Recent improvements of CBS, like Weighted Dependency Graph (WDG), Prioritizing Conflicts, etc., have also been adapted to EECBS, which makes it run significantly faster than the state-of-the-art bounded-suboptimal MAPF algorithms on a variety of MAPF instances.

To convert this to a non-holonomic agent problem, we included a lattice-based planner with motion primitives as the low-level planner. Motion primitives are predefined paths with a particular length and change in orientation. We used 10 motion primitives to define possible neighboring states from the current state. These primitives can be seen in the image below,

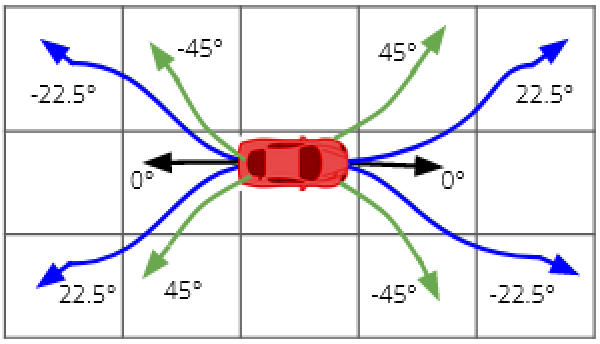


Fig. 2: Motion primitives generated

As can be observed from Fig. 2, there are 5 forward motion primitives and 5 reverse motion primitives. The angles written beside each primitive represent the amount and direction of the change in orientation for that primitive with respect to the initial orientation. The length of the paths for the shorter primitives is considered to be 1, and for the longer primitives, is considered to be √5.

The exact paths for the primitives are calculated using bezier curves. This was done keeping in mind that the paths need to be tangent at the start and end points with respect to the corresponding orientations for continuity. The control points are chosen in a manner that can be seen in Fig. 3 below.

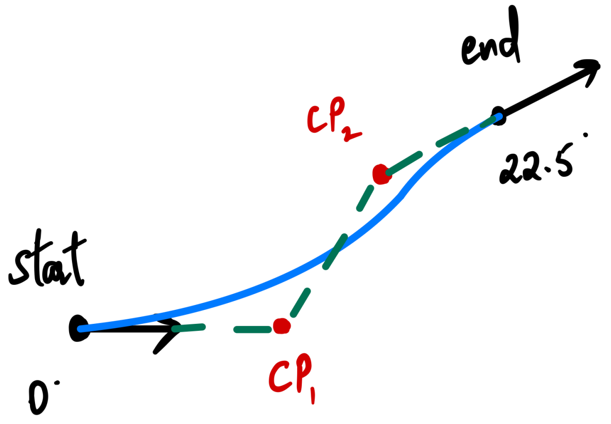


Fig. 3: Path generated using Bezier curve

The first control point is along the initial orientation direction and 1 unit away. The second control point is in the direction opposite to the final orientation and a distance 1 unit away. Once this is done, we traverse the path in small increments and, at each step, see if the cell index changed with respect to the previous location. This gives us cell crossovers in the motion primitive. Each cell crossover is designed to take 1 second, so the number of cell transitions defines the duration of the motion primitive. Each cell transition is checked for both collisions and constraints. If all transitions satisfy this, the primitive is counted as a valid neighbor.

EECBS was originally designed for a 4-connected grid with an (x, y) state space. We decided to extend the state space to (x, y, theta) to make it more applicable. So another major task was modifying the conflict detection and adding additional constraints. We had to restrict diagonal movements between obstacles and detect collisions when agents move orthogonally to each other, as shown in Fig 4 below.

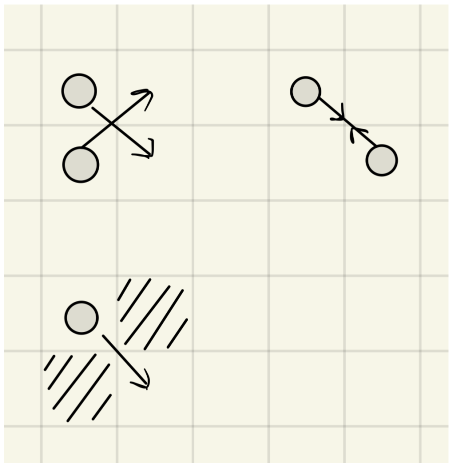


Fig. 4: Diagonal Edge Conflicts

Our start locations for each robot are a set of (x, y, theta) points. To simplify the problem, we only consider (x, y) in the goal configuration so that the robot can reach the goal in any orientation. The high-level planner's objective is to minimize the objective function, which is the sum of the path lengths of all the agents. For the low-level planner, the heuristic we used was backward Dijkstra but with the same motion primitives.

Fig. 5(a) shows how EES works under the hood, and Fig. 5(b) shows how EES is integrated into CBS to give the resultant EECBS.

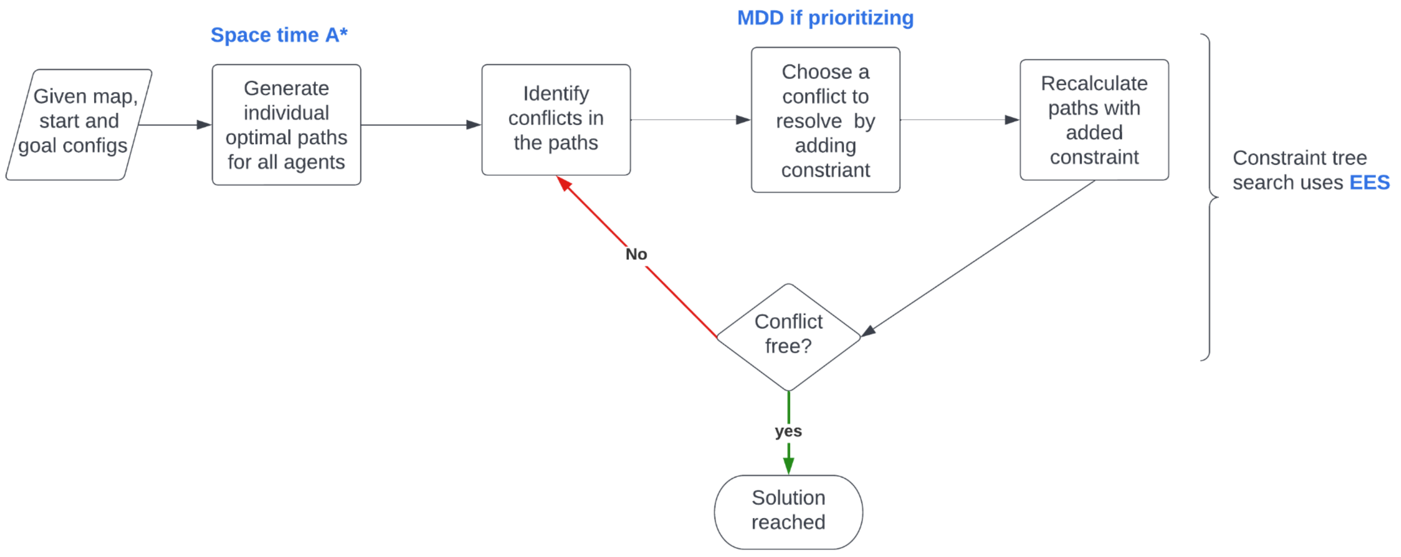
```
$selectNode$
1.  **if** $\widehat{f}(best_{\widehat{d}}) \leq w \cdot f(best_f)$ **then** $best_{\widehat{d}}$
2.  **else if** $\widehat{f}(best_{\widehat{f}}) \leq w \cdot f(best_f)$ **then** $best_{\widehat{f}}$
3.  **else** $best_f$
```

Best f: Regular A* open list node with lowest f value

Best f-hat: Inadmissible cost function - estimating the cost-to-go under a constraint tree node

Best d: Node with minimum distance-to-go - Focal sorted based on this

Fig. 5(a): Explicit Estimation Search Algorithm



Space time A*
MDD if prioritizing
Given map, start and goal configs
Generate individual optimal paths for all agents
Identify conflicts in the paths
Choose a conflict to resolve by adding constraint
Recalculate paths with added constraint
Constraint tree search uses EES
No
Conflict free?
yes
Solution reached

Fig. 5(b): EECBS Algorithm

Experiment and Results

All experiments run the EECBS approach extended to (x,y,theta) state space along with WDG for high-level heuristics and prioritization of conflicts. All agents are default initialized with theta zero at the start position, and the suboptimality factor used is 1.2.

We first test our approach on a simple 5X5 map with two agents discussed in class. The start and goal positions of the agents are shown in Fig. 6 first image. The individual optimal paths of the agents have vertex and edge conflicts. However, these are detected and resolved to produce the final conflict-free paths, as shown in the sequence in Fig. 6.

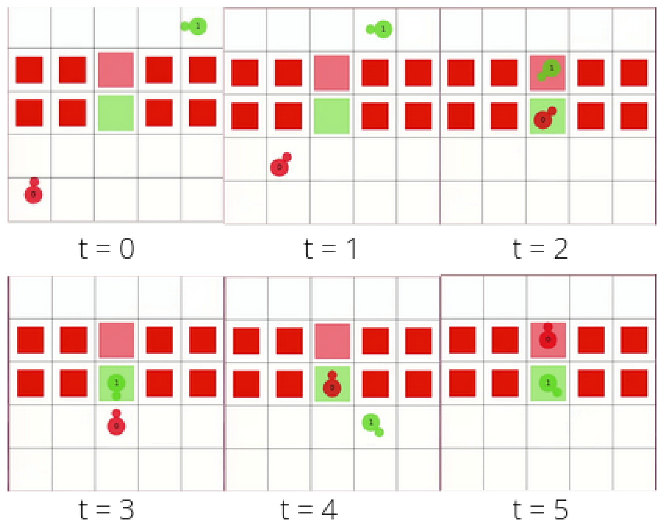


Fig. 6: Example one test setting and conflict-free path

Similarly, we test our approach on a 4X4 map with two agents to demonstrate the agents avoiding diagonal conflicts. Consider the problem shown in Fig. 1 with a diagonal conflict in the initial individual optimal paths. Our approach resolves this, and the final conflict-free paths obtained are shown in Fig. 7.

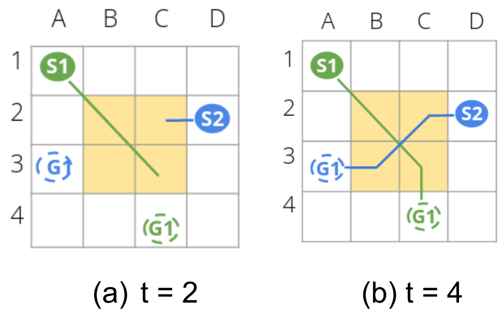


Fig. 7: Conflict-free paths for example two. Agent s2 waits one time step to avoid collision

We next tested our approach on 5 different maps with 25 random scenarios for 8 agents taken from the MAPF benchmark **[6]** with a time limit of 60 sec. The size of these maps varies from 32X32 to 128X128. Note, for these experiments alone; we did not use prioritization of conflicts. Our approach found a feasible solution for all but 2 of the tests (shown as zero runtime on the map), achieving a success rate of 98.4%. The runtime for each of these tests is shown in Fig. 8. We observe that the runtime generally increases with the size of the map.

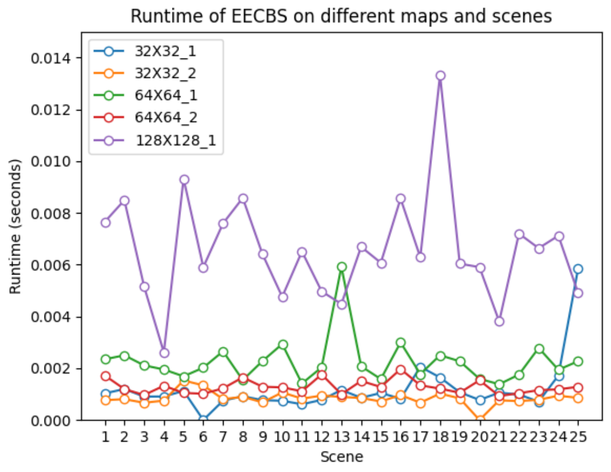


Fig. 8: Runtime of EECBS on different maps and scenarios

Further, to study the advantage of using longer primitives, we repeated the above experiment with the six short primitives alone, and the results were almost identical. Omitting longer primitives leads to a decrease of 5% in runtime on average over all the maps. However, without longer primitives, the path cost on average shows a marginal increase of 4.4% except on map 3, this is shown in Fig. 9.

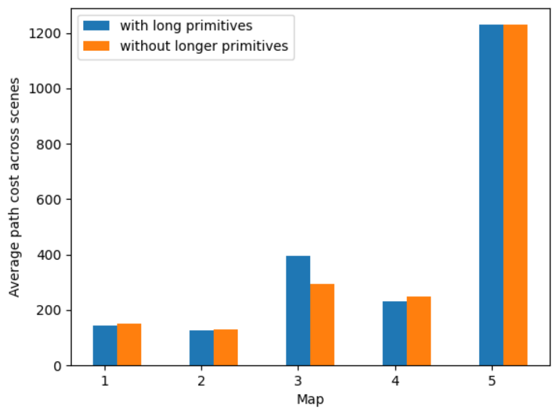


Fig. 9: Change in path cost when planning with and without longer primitives

We next tested our approach on random_map_32_32_20 from the MAPF benchmark with 10 agents with different sub-optimality factors ranging from 1.02 to 1.5 and a time limit of 60 sec. We noted that the planner could not solve with a suboptimality bound of less than 1.1 within the set time limit. The runtime for the instances when it was solved is shown in Fig. 10.

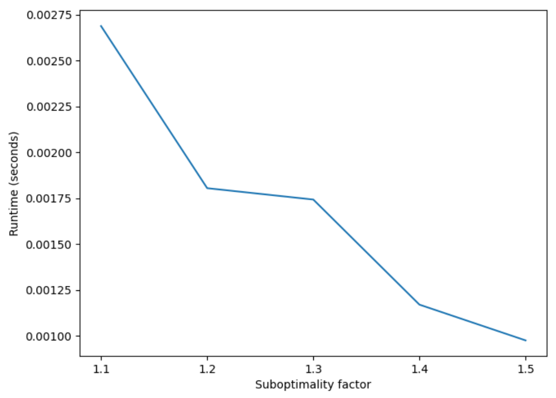


Fig. 10: Variation of runtime on random_map_32_32_20 with 8 agents and varying suboptimality factor

The visualization for the example shown in Fig. 6 and a test on random_map_32_32_20 from the MAPF benchmark with 10 agents and a suboptimality bound of 1.2 can be found in demo_link.

Conclusion

We successfully extended EECBS to x, y, theta state space and incorporated a lattice planner to evaluate more realistic scenarios. We modified the Weighted Dependency Graph (WDG) and Prioritizing Conflicts to work in this new state space. Finally, we evaluated our work on multiple standard (and customized) MAPF instances. Multiple extensions to our work can be done as future work. We can try incorporating even longer primitives, do away with point robot assumption, and change costs for different primitives, especially backward motions.

Work breakdown

These were just the sections people were made responsible for; the team often worked collaboratively, helping each other through roadblocks.

1. Primitives Generation - Manuj Trehan
2. Conflict detection and constraint addition - Akshaya KS
3. WDG and prioritization of conflicts - Harshit Agrawal
4. Visualization - Vrushali Patil
5. Integration and testing - Nevin Valsaraj

References

[1] Sharon, G., Stern, R., Felner, A., & Sturtevant, N. (2021). Conflict-Based Search For Optimal Multi-Agent Path Finding. *Proceedings of the AAAI Conference on Artificial Intelligence*, *26*(1), 563-569. https://doi.org/10.1609/aaai.v26i1.8140

[2] Barer, Max et al. “Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem.” *Symposium on Combinatorial Search* (2014).

[3] Li, Jiaoyang et al. “EECBS: A Bounded-Suboptimal Search for Multi-Agent Path Finding.” *AAAI Conference on Artificial Intelligence* (2020).

[4] Thayer, Jordan Tyler and Wheeler Ruml. “Bounded Suboptimal Search: A Direct Approach Using Inadmissible Estimates.” *International Joint Conference on Artificial Intelligence* (2011).

[5] Yu, Jingjin & LaValle, S.M.. (2013). Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. Proceedings of the 27th AAAI Conference on Artificial Intelligence, AAAI 2013. 27. 1443-1449. 10.1609/aaai.v27i1.8541.

[6] Stern, Roni & Sturtevant, Nathan & Felner, Ariel & Koenig, Sven & Ma, Hang & Walker, Thayne & Li, Jiaoyang & Atzmon, Dor & Cohen, Liron & Kumar, T. & Boyarski, Eli & Barták, Roman. (2019). Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks.