Multi-Agent Path Planning With Dynamic Obstacles in a Partially Known Environment

Rishab Shah
Boston University
8 St. Mary's Street Boston MA 02215
rishah@bu.edu

Abstract

We propose an efficient variant of the D lite algorithm for planning in partially known maps and environments with dynamic obstacles and extend it to a multi-robot system. We make this into a motion planning problem rather than a coverage problem which is common with multi-agent systems. We find ourselves at a crossroad in selecting the mode of communication between the robots with a centralized or a decentralized, distributed approach. A distributed approach is used for communication between the agents as it allows for lower complexity and better planning.*
A simulation scenario reflects the experimentation for this problem. The results are evaluated based on optimality, which means shortest path with no collisions.

1. Introduction

Robots operating in an unknown, partially known and/or dynamic environment that are required to perform a given task are faced with the problem of deciding where to go to get the most relevant information for their current task. We are particularly interested in the simplest task in a path planning problem, i.e., finding an optimal path for all the robots between two locations.

When the environment is known, the path planning problem consists of computing paths that allow the robots to sense the regions of interest in an optimal way according to some metric. However, when the environment is unknown or partially known, the robots must also learn the structure of the environment by identifying which regions are interesting and which are not. Hence, these paths are where the most sensory information can be obtained by the robots.

Given a directed or undirected graph and a set of agents with unique start and goal vertices, the Multi-Agent Path Finding (MAPF) problem is to find collision-free paths for all agents from their respective start vertices to their respective goal vertices [3]. Minimizing the solution cost given by the sum of the travel times of the agents along their paths is NP-hard [23].

Algorithms that are commonly used in partially known environments with dynamic obstacles are D* Lite and variants like Focused D*, Moving Target D* Lite (which is primarily used for tasks with moving targets). The actual performance of D* Lite is dependent upon the size of the grid, number of node expansions and heap sorting. D* Lite is free from the drawbacks like converging to local minimums as is the case with Potential-field algorithms.

In this paper, we present a novel implementation of the D* Lite algorithm applied to a multi-robot system as they traverse the environment towards their goal. To consider all possibilities, we look at the case when the robots have different start and end locations. We exploit the fact that the robots have a limited field of view around them, and of the obstacles that every other robot update their map with as they move toward the goal.

2. Background and Related work

2.1. Different applications

Multi-agent systems are often adopted to accomplish tasks where cooperation between different robots is preferred, like situations that are hazardous or time and power consuming for a single agent. Extensive work has already been done [4] [21] [25] [19] in the application areas such as , factory floor robots, area reconnaissance, task reassignment in multi-robot teams, cleaning robots etc. Traditional path planning methods such as Visibility graph, Free space method, Grid method, Topological method and Potential-field method offer great success in multi-robot path planning.

Path finding is also a component of many video games. For example, agents in turn-based or real-time strategy games need to plan collision-free paths from their current locations to their goal locations, often in dynamic and congested environments. It is easier for players to control hundreds of agents by moving the agents in a team rather than individually. MAPF can be solved via reductions to other well-studied problems [18] [22] [6] or by optimal or suboptimal MAPF algorithms [16] [20]. Many MAPF algorithms have been used on maps from video games [14]. MAPF algorithms focus on the aspect of collision avoidance and build experience graphs [13] that are modelled in the form of human-generated directed highways whereas this paper discusses methods to find the optimal paths for a multi agent system.

Multi-agent systems are also employed in both the military and the civilian domain for surveillance missions with a system of multiple Unmanned Aircraft by means of a Kalman Filter technique [8]. They are slightly different from the strategy we plan to use as they are based on the development of a target tracking algorithm to provide information on both target and drones motion on the surveillance area by means of a Kalman Filter and Bayesian network.

Search and Rescue operations are also performed by adopting the multi-agent systems and they may be generally characterized through multiple dimensions and attributes including: stationary vs. moving target search, discrete vs. continuous time and space search, static/dynamic as well as open and closed-loop decision models [2]. The difference between the aforementioned and the algorithm discussed in this paper is that this is a low complexity algorithm that aims to take care of the problems faced in navigation of multi robot systems.

2.2. Path planning algorithms

Focused D* [17], and D*Lite [9] are popular replanning algorithms amenable for path-planning in partially known terrain. D*Lite extensions are used in the navigation algorithms for a Mars Rover [7], and in the DARPA Urban challenge competition [11]. [Making A* Run Faster than D*-Lite for Path-Planning in Partially Known Terrain]

When the agent has partial knowledge of the terrain we make a free space assumption [24] which is an optimistic assumption about the grid in which unknown cells are considered obstacle-free. Adaptive A* (AA*) [10] is a replanning algorithm that uses A* to compute a complete solution each time one is needed.

In applications where robots have to continually monitor or sense regions of interest for a long period of time, the robots generate closed paths that allow them to sense regions of interest. These paths are referred as Interesting Closed Paths (IC paths). The robots could then travel these closed paths for long periods of time until the monitoring task is done. A wide range of applications require robots to move along IC paths in unknown environments in order to concentrate the robots resources in the regions of interest and avoid wasting them in non-interesting regions. The algorithm presented in [15] allows the user to have liberty of controlling the robots speed along the IC paths.

2.3. Planning under uncertainty

Another paper [12] uses the uncertainty of the mapped obstacles in the planning stage to minimize the chance of collision when navigating. This method fundamentally bases on Rapidly-Exploring Random Trees (RRTs) and inherits the sampling-based planners sub-optimality and probabilistic completeness.

Newer approaches like [5] [1] involve stochastic robot observations and dynamics: after applying a motion command, the actual robot position differs from the predicted one, due to actuation noise. Even more important, the robot does not know, a priori, which measurements will be acquired and what the (future) sensor readings will be.

3. Problem Definition

3.1. Description of the problem

We assume that a single obstacle map is available for all the agents involved in performing the given task. The map is available in shape of a graph $G = (V, O, c)$, where V is the set of vertices, O is the set of arcs and c is a cost function $c : O \rightarrow \mathbb{R}$. This graph represents the connectivity of both free and occupied space around multiple robots involved.

We define a vertex within graph G as $V_{x,y} = (r_{id}, t)$. Here x and y are the index of the vertex on grid-like eight connected graph and R_{id} is the robot id. Variable is set to null (ϕ) for some vertex that is not part of a planned path for any of the robots. We assume that a set consists of robot identities and is defined by $R = \{r_1, r_2, \ldots r_k\}$ represents k robots (agents) which share the same obstacle map. We also assume that collisions occur when a robot r_i tries to enter a vertex which has the value of its $t \neq 0$. Each robot should ideally follow a collision-free path in order to reach its predefined goal. But if for example, vertex $v_{x,y}$ happens to be a part of the path for a robot, the corresponding R_{id} is added.We define t to the obstacle map, as the remaining time for a vertex to be not occluded again. This concept of *tenancy* enables us to sample paths into discrete space-time segments consequently allowing the time-sharing of a single vertex for more than one robot path. Thus when $t = 0$, this condition switches the state of vertex v from occupied to unoccupied.

3.2. Assumptions

The robots from set R have a predefined set of goals $E = \{e_1, e_2, \ldots e_k\}$ with each of their start positions also defined as $S = \{s_1, s_2, \ldots s_k\}$. A limitation of the D* Lite algorithm is that the positions of the goal cannot be changed until all robots have reached their destinations. In this paper, we assume that each robot perceives any other robot as a dynamic obstacle and not to take advantage of the spatiotemporal nature of obstacles as all robots share the memory and know the path of the other robots.

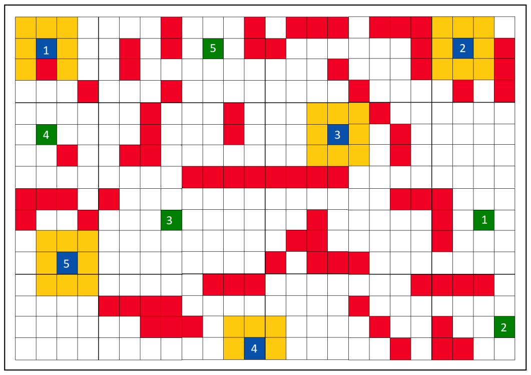

Figure 1. The obstacle map (of size 24x16) at $t = 0$, the red blocks are obstacles, blue blocks are the start locations of the robots (numbered), green blocks are the goal locations, and the yellow blocks are the visibility of each of the robots.

Now the algorithm is expected to compute feasible collision-free paths for all the robots defined by set R. All robot paths must start at start positions defined by set S and end at goal positions defined by set E.

A heuristic function g for a path-planning problem is a non-negative function such that *g(s)* estimates the cost of a path from s_i to e_i . Function g is admissible if for every state s_i, $g(s_i)$ does not overestimate the cost of any path from s_i to e_i . Furthermore, we say g is consistent if for every *(s, t)* $\in O$ it holds that $g(s_i)c(s_i,t)+g(t)$, and $g(e_i) = 0$. $h(s)$ is the path to goal, if the robot moves after planning and then detects cost changes again, then the constants need to get added up. We do this in the variable k_m, and U is the priority queue.

The robots also have limited visibility around themselves and can sense only around their neighborhood (here, right connected graph).

4. The path planner

4.1. The obstacle map

For a multi-agent system, each robot needs to communicate to each other their start positions and the information in their field of view. We assume a partially known map, and hence the robots are aware of the goal location. We use a shared memory that acts as a central server for sharing the information about the map and updating the path. We assume all the robots move at the same, constant speed. The algorithm accepts the obstacle map with or without indication of static or dynamic obstacles. Static or dynamic obstacles can be added during the algorithm execution as and when visible to the participating robots. A vertex can be declared as a static obstacle by a change to the value of tenancy t as shown in the following cases. If the *tenancy* of the vertex $v_{x,y}$ is finite, then:

$$\begin{aligned} set\ tenancy(v_{x,y}) &= \infty; \\ set\ cost(Predecessor(v_{x,y}), v_{x,y}) &= \infty; \end{aligned} \tag{1}$$

A vertex can be set as unoccupied by checking whether the *tenancy* of the vertex $v_{x,y}$ is valid (>0), then:

$$\begin{aligned} set\ tenancy(v_{x,y}) &= 0; \\ set\ R_{id} &= \phi; \\ set\ cost(Predecessor(v_{x,y}), v_{x,y}) &= 1; \end{aligned} \tag{2}$$

A vertex can be declared as a dynamic obstacle (for which the tenancy is known) if the *tenancy* of the vertex $v_{x,y}$ is 0, then:

$$\begin{aligned} set\ tenancy(v_{x,y}) &= t; \\ set\ R_{id} &= i; \\ set\ cost(Predecessor(v_{x,y}), v_{x,y}) &= \infty; \end{aligned} \tag{3}$$

It must also be ensured that a vertex has to be unoccupied prior to declaring it as a dynamic obstacle. Currently, the algorithm perceives any other robots path as a dynamic obstacle. Now the algorithm is expected to compute feasible collision-free paths for all the robots defined by set R.

4.2. Algorithm

4.3. Multi agent D* Lite

```
1:  **procedure** initialize()
2:  set Robots_array = R;
3:  G = initialize_obstacle_map(S,E);
4:  **procedure** Multi agent D* Lite
5:  prioritize robot list(g,R);
5:  **for** $r_i \in R$ **do**
6:  new D* Lite($r_i$);
6:  **end for**
7:  **procedure** update_obstacles(graph G)
7:  **for** $v_{x,y} \in V$ where $tenancy(v_{x,y}) \neq 0$ or $\infty$ decrement($tenancy(v_{x,y})$); **do**
8:  **if** any vertices are found where $tenancy(v_{x,y}) = 0$ **then**
9:      $set\ R_{id} = \phi$;
10:     $set\ cost(Predecessor(v_{x,y}), v_{x,y}) = 1$;
11: **end if**
12: **if** any vertices are found where $tenancy(v_{x,y}) = \infty$ **then**
13:     $set\ cost(Predecessor(v_{x,y}), v_{x,y}) = \infty$;
14: **end if**
14: **end for**
15: **procedure** update R
16: $set\ R = \{\phi\}$
17: update_obstacles($v_{x,y}$);
17: **for** all vertices $v_{x,y} \in V$ affected by obstacle change **do**
18: $R.add(r_{id}(v_{x,y}))$;
18: **end for**
```

```
19: procedure main
20: initialize();
21: while true do
22:     Multi agent D* Lite(R, G);
23:     update R(R, G);
24:     update(S);
25: end while=0
```

4.4. New D* Lite

```
1: procedure CalculateKey(s)
2: $return\ [min(g(s), rhs(s)) + h(s_i, s) + k_m; min(g(s), rhs(s))]$;
3: procedure Initialize()
4: $U = \phi$;
5: $k_m = 0$;
5: for $all\ s \in S$ do
6: $rhs(s) = g(s) = \infty$;
6: end for
7: $rhs(e_i) = 0$;
8: $U.Insert(e_i, [h(s_i, e_i); 0]$;
9: procedure UpdateVertex(v)
10: if $v \neq e_i$ then
11:     $rhs(v) = \min_{s' \in Succ(v)} (c(v, s') + g(s'))$;
12: else if $v \in U$ then
13:     $U.Remove(v)$;
14: else if $((g(v) \neq rhs(v)) AND (tenancy(v) = 0))$ then
15:     $U.insert(u, CalculateKey(v))$;
16: end if=0
```

```
1: procedure ComputeShortestPath()
2: while $U > TopKey() < CalculateKey(s_i)\ OR\ rhs(s_i) > g(s_i)$ do
3:     $v = U.TopKey()$;
4:     $k_{old} = U.TopKey()$;
5:     $k_{new} = CalculateKey(v)$;
6:     if $k_{old} < k_{new}$ then
7:         $U.Update(v, k_{new})$;
8:     else if $g(v) > rhs(v)$ then
9:         $g(v) = rhs(v)$;
10:        $U.Remove(v)$;
11:    end if
11:    for $all\ s \in Pred(v)$ do
12:    if $s \neq e_i$ then
13:        $rhs(s) = min(rhs(s), c(s, v) + g(v)$;
14:        $UpdateVertex(s)$;
15:    else
16:        $g_{old} = g(v)$;
17:        $g(v) = \infty$;
18:    end if
18:    end for
18:    for $all\ s \in Pred(v) \cup \{v\}$ do
19:    if $rhs(s) = c(s, v) + g_{old}$ then
20:        if $s \neq e_i$ then
21:            $rhs(s) = \min_{s' \in Succ(s_i)} (c(s, s') + g(s'))$;
22:        end if
23:    end if
24:    $UpdateVertex(s)$;
24:    end for
25: end while
26: procedure Main()
27: $e_i = s_i$;
28: $wait\ for\ signal\ from\ Multi\ agent\ D^*\ Lite$
29: $and\ then\ scan\ for\ changed\ edge\ costs$;
30: $ComputeShortestPath()$;
31: $s = s_i$; $t = 1$;
32: while $s \neq e_i$ do
33:     $settenancy(s) = t$;
34:     $s = \min_{s' \in Succ(s)} (c(s, s') + g(s'))$;
35:     $robot_i d(s) = r_i$;
36:     $t++$;
37: end while=0
```

The algorithm chooses a priority order for the robots based on a heuristic criterion. Based upon this choice, the server applies the new D-star Lite algorithm one by one to calculate shortest collision-free path for each robot. In this sequential execution of D-star Lite algorithm, a path is planned for each robot while taking into account the paths that have already been generated for previous robots. During this sequential execution, each time a path is generated for any given robot, the algorithm treats all the nodes involved in previously generated paths as obstacles until the tenancy (t) decrements to value of zero. After generation of shortest path, the new D*-Lite algorithm also attaches robot ID information to each vertex involved in robot path. The algorithm also updates tenancy values with each vertex in robot path with a 1 second increment to each vertex as we move from start vertex to the goal vertex. Since the execution time of D* Lite is dependent upon the frequency of change in obstacles, multi-robot D* Lite adds to the time complexity of D* Lite by a polynomial factor. This factor is directly proportional to the number of robots (k) involved in multi-robot path planning. The factor is also dependent upon the probability of path cross over and probability of obstacle change.

The images show the different cases of running the algorithm. Fig. (2) shows the path when no obstacles are changed in their path. The blocks A and B are highlighted to show that the other robot's path occludes its original shortest path and it adjusts accordingly. In block B, there is no occlusion when the robots are moving and hence they continue along their original shortest path. Fig. (3) shows when additional obstacles are introduced in the obstacle map.

We use Mobile Robot Programming Toolkit (MRPT) to run the algorithm. The units of occupancy grid are mapped to nodes in the eight-connected obstacle-map G, thus con-

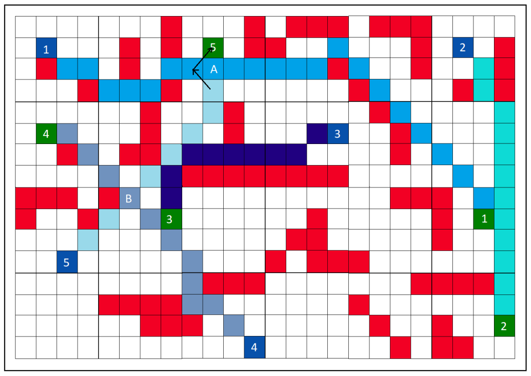

Figure 2. The obstacle map (of size 24x16) at $t = 0$, the red blocks are obstacles, blue blocks are the start locations of the robots (numbered), green blocks are the goal locations, and the yellow blocks are the visibility of each of the robots and they have their paths shown.

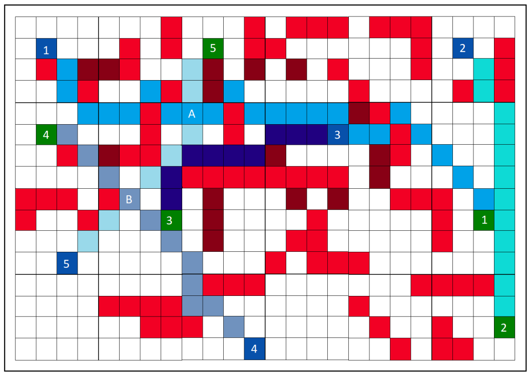

Figure 3. The obstacle map (of size 24x16) at $t = 0$, the red blocks are obstacles, blue blocks are the start locations of the robots (numbered), green blocks are the goal locations, the yellow blocks are the visibility of each of the robots, and new obstacles are shown in dark red.

verting physical adjacency relationships to graph connectivity. We use the minimum Euclidean distance to be the priority heuristic. The obstacles are also assumed to move at a constant speed randomly and can also appear without existing before. We also tried to increase the number of robots to 10 and increase the obstacle map to 30x30, and the results are degraded in the sense that they do not find any path to the goal when the number of robots are more than 5, while the increase in resolution just increases the time-complexity of the algorithm and it runs slower.

4.5. Conclusion

We introduced a new version of the D* Lite algorithm. It maintains the capability to efficiently remove collisions due to multiple robot path overlapping. The inherent robustness of D* Lite algorithm makes its multi-robot version tolerant to inaccuracies in map. The algorithm has shown degraded results whenever number of robots is increased, yet for a small group of robots, the algorithm contends to be a feasible choice.

References

[1] T. Bailey and H. Durrant-Whyte. Simultaneous localization and mapping (slam): Part ii. *IEEE Robotics & Automation Magazine*, 13(3):108–117, 2006.

[2] J. Berger and N. Lo. An innovative multi-agent search-and-rescue path planning approach. *Computers & Operations Research*, 53:24–31, 2015.

[3] L. Cohen, T. Uras, T. K. S. Kumar, H. Xu, N. Ayanian, and S. Koenig. Improved solvers for bounded-suboptimal multi-agent path finding. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 3067–3074. AAAI Press, 2016.

[4] R. N. De Carvalho, H. Vidal, P. Vieira, and M. Ribeiro. Complete coverage path planning and guidance for cleaning robots. In *Industrial Electronics, 1997. ISIE'97., Proceedings of the IEEE International Symposium on*, volume 2, pages 677–682. IEEE, 1997.

[5] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[6] E. Erdem, D. G. Kisa, U. Öztok, and P. Schüller. A general formal framework for pathfinding problems with multiple agents. In *AAAI*, 2013.

[7] D. Ferguson and A. Stentz. Field d*: An interpolation-based path planner and replanner. *Robotics research*, pages 239–253, 2007.

[8] D. Gentilini, N. Farina, E. Franco, A. E. Tirri, D. Accardo, R. S. L. Moriello, and L. Angrisani. Multi agent path planning strategies based on kalman filter for surveillance missions. In *Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI), 2016 IEEE 2nd International Forum on*, pages 1–6. IEEE, 2016.

[9] S. Koenig and M. Likhachev. Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics*, 21(3):354–363, 2005.

[10] S. Koenig and M. Likhachev. A new principle for incremental heuristic search: Theoretical results. In *ICAPS*, pages 402–405, 2006.

[11] M. Likhachev, D. I. Ferguson, G. J. Gordon, A. Stentz, and S. Thrun. Anytime dynamic a*: An anytime, replanning algorithm. In *ICAPS*, pages 262–271, 2005.

[12] K. Ok, S. Ansari, B. Gallagher, W. Sica, F. Dellaert, and M. Stilman. Path planning with uncertainty: Voronoi uncertainty fields. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 4596–4601. IEEE, 2013.

[13] M. Phillips, B. J. Cohen, S. Chitta, and M. Likhachev. E-graphs: Bootstrapping planning with experience graphs. In *Robotics: Science and Systems*, volume 5, 2012.

[14] D. Silver. Cooperative pathfinding. *AIIDE*, 1:117–122, 2005.

[15] D. E. Soltero, M. Schwager, and D. Rus. Decentralized path planning for coverage tasks using gradient descent adaptive control. *The International Journal of Robotics Research*, 33(3):401–425, 2014.

[16] T. Standley and R. Korf. Complete algorithms for cooperative pathfinding problems. In *IJCAI*, pages 668–673, 2011.

[17] A. Stentz et al. The focussed d^* algorithm for real-time replanning. In *IJCAI*, volume 95, pages 1652–1659, 1995.

[18] P. Surynek. Reduced time-expansion graphs and goal decomposition for solving cooperative path finding sub-optimally. In *IJCAI*, pages 1916–1922, 2015.

[19] M. Takahashi, T. Suzuki, H. Shitamoto, T. Moriguchi, and K. Yoshida. Developing a mobile robot for transport applications in the hospital domain. *Robotics and Autonomous Systems*, 58(7):889–899, 2010.

[20] K.-H. C. Wang and A. Botea. Mapp: a scalable multi-agent path planning algorithm with tractability and completeness guarantees. *Journal of Artificial Intelligence Research*, 42:55–90, 2011.

[21] E. K. Xidias, A. C. Nearchou, and N. A. Aspragathos. Vehicle scheduling in 2d shop floor environments. *Industrial Robot: An International Journal*, 36(2):176–183, 2009.

[22] J. Yu and S. M. LaValle. Planning optimal paths for multiple robots on graphs. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3612–3617. IEEE, 2013.

[23] J. Yu and S. M. LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In *AAAI*, 2013.

[24] A. Zelinsky. A mobile robot exploration algorithm. *IEEE Transactions on Robotics and Automation*, 8(6):707–717, 1992.

[25] R. Zlot and A. Stentz. Market-based multirobot coordination using task abstraction. In *Field and Service Robotics*, pages 167–177. Springer, 2003.