Khalid Al-Mutib, Mansour AlSulaiman, Muhammad Emaduddin, Hedjar Ramdane, Ebrahim Mattar / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622
www.ijera.com Vol. 2, Issue 2,Mar-Apr 2012, pp.1414-1419

D* Lite Based Real-Time Multi-Agent Path Planning in Dynamic Environments

Khalid Al-Mutib, Mansour AlSulaiman, Muhammad Emaduddin, Hedjar Ramdane
Dept. of Computer Engineering
College of Computer Science and Information
King Saud University, Riyadh, Saudi Arabia

Ebrahim Mattar
Electrical & Electronics Engineering Dept.
College of Engineering, University of Bahrain
Kingdom of Bahrain

Abstract
D* based navigation algorithms provide robust and real-time means of achieving path planning in dynamic environments. Author of this paper introduces a notion of predictable time-based obstacles. The algorithm proposed in the paper defines a centralized obstacle-map that is shared among multiple agents (robots) performing path planning. Each robot plans its path individually on an obstacle-map using a slightly modified version of D* Lite and then shares an updated version of the map, which includes its planned path as a new obstacle, with its peers. The planned paths appear as temporary time-based obstacles to peer robots. Planned paths are divided into discrete temporal sections so as to help peer robots optimize paths temporally. The proposed algorithm also presents a priority measure which helps us decide the optimized sequence of individual path-planning order followed by cooperating robots. Since the implemented algorithm is tested in simulation using Mobile robot Programming Toolkit, the Real–time performance analysis is done to confirm the real-time execution time of the proposed algorithm.

Keywords- robotics; path-planning; D*; navigation; multi-agent systems

1. INTRODUCTION
Multi-robot path planning has huge number of applications especially whenever the problem requires teamwork from robots. Extensive work has already been done [1][2][3][4][5] on application of multi-robot path-planning in the application areas such as cleaning robots, factory floor robots, area reconnaissance, hospital transport robots, task reassignment in multi-robot teams 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 but performance and execution optimization have always been a problem to deal with[6][7][9].

The original single robot dynamic path planning algorithm D* Lite by Koenig [8] is a hardware tested and proven real-time algorithm. The actual performance of D* Lite is dependent upon the size of the grid, number of node expansions and heap sorting as referred in [8]. Our proposed algorithm is real time since path planning for an individual robot is done by employing D* Lite algorithm. D* Lite algorithm is better suited towards navigation in inaccurate environment and also is free from the pitfalls like converging to local minimums as is the case with Potential-field algorithms. It may be noted here that since path planning suggested by our algorithm is carried out sequentially by participating robots, thus it will introduce a delay before each robot is able to start travelling on a planned path. This delay depends on the ranking of any robot in the prioritization done by our proposed algorithm. In case of a change in the obstacle-map, additional delay can occur for the robots for which the path needs to be re-planned. This approach greatly differs with the multi-level trajectory planning approach used by Berg & Overmars [10]. Berg & Overmars approach uses a grid based road-map where paths are discretized on the basis of state and time.

Our approach uses the same model to define the problem but path planning and path reconfiguration is more robust since Berg & Overmars [10] require the motions of the obstacles to be known before-hand. We use D-star Lite for real-time path-planning of individual robots. In case path reconfiguration is required due to a change in obstacle map, we use D-star Lite for the affected paths. This approach gives us ability to plan paths within inaccurate environments such as SLAM problems and among obstacles that have unpredictable motion and footprint. Frequent changes in obstacle map can lead to computation intensive execution but such an approach adds to the reliability of the algorithm in environment crowded with obstacles.

2. PROBLEM DESCRIPTION
2.1 Obstacle Map
We assume that a single obstacle map is available for all robots participating in the problem. The map is available in shape of a graph $G = (V, E)$. This graph represents the connectivity of both free and occupied space around multiple

robots involved in our problem definition. The dimensions of the real world space represented by each vertex in the graph is a tunable parameter as it factors-in the physical constraints presented by real-world navigation problems.

2.2 Occupancy-life

We define a vertex within graph G as $V_{x,y} = (robot\_id, t)$. Here x and y are the index of the vertex on grid-like eight connected graph. Variable $robot\_id$ is set to *null* ($\emptyset$) for some vertex $v_{x,y}$ that is not part of a planned path for any of the robots. But if for example, vertex $v_{x,y}$ happens to be a part of the path for a robot, the corresponding robot ID is added to the $robot\_id$ variable. We attach the notion of time to the already provided obstacle map by defining t as the remaining time (in milliseconds) for a vertex to be unoccupied (accessible) again. This concept of *occupancy life* 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.

2.3 Agents and Paths

We assume that a set consists of robot identities and is defined by $R = \{r_1, \dots, r_k\}$ 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 sequence of vertices for example, $c_1 = \{v_{0,0}, v_{1,0}, v_{2,1}, v_{2,2}\}$, in order to reach its pre-defined goal. We will refer to c_i as a collision-free path for a single robot with ID *i*. The obstacles in the obstacle map can be sensed by all robots through a Boolean function $B(v_{x,y})$ where $v_{x,y}$ denotes the particular node which was analyzed for an obstacle by a robot.

2.4 The Problem

The robots from set R have pre-defined set of goals $D = \{g_1, \dots, g_k\}$ where g_1 indicates goal for robot r_1 and so on. The robots also have their initial start positions defined by set $S = \{s_1, \dots, s_k\}$. The predicate $(s_u \neq s_v)\&\&(g_u \neq gv\ given\ u \neq v$ holds true for both sets D and S. We currently confine ourselves to a 2D physical workspace for a purpose of analysis in this paper. It is worthwhile to mention here that the same algorithm is extensible to 3D physical workspace for robots. The sets C, D and S are illustrated in Fig.1 via a basic example.

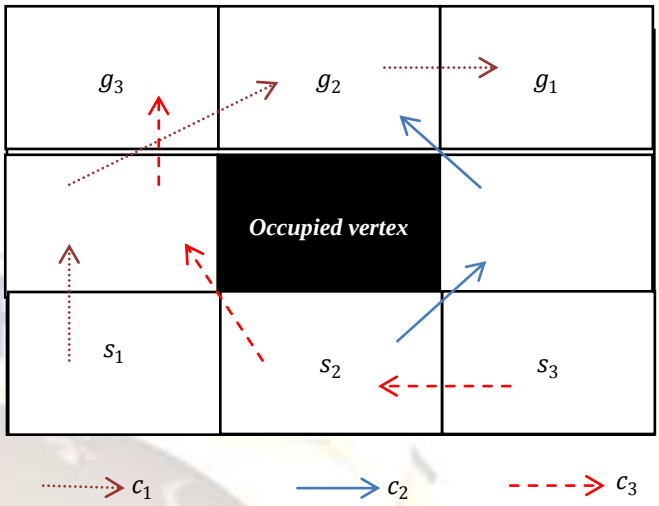

Fig. 1. Basic example of physical space represented by graph G

The obstacle-map G must indicate start *(S)* and goal *(D)* positions where goal positions cannot be changed until all robots reach their goal locations since this is a limitation put forward by D* Lite algorithm. 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 occupancy-life t as follows.

$$if\ (occupancy\_life(v_{x,y}) \neq \infty) \quad \dots (1)$$
$$set\ occupancy\_life(v_{x,y}) = \infty;$$
$$cost(Predecessor(v_{x,y}), v_{x,y}) = \infty;$$

A vertex can be declared as unoccupied by executing the following.

$$if\ (occupancy\_life(v_{x,y}) > 0) \quad \dots (2)$$
$$set\ occupancy\_life(v_{x,y}) = 0;$$
$$set\ robot\_id(v_{x,y}) = \emptyset;$$
$$cost(Predecessor(v_{x,y}), v_{x,y}) = 1;$$

A vertex can be declared as a dynamic obstacle (for which the occupancy-life is known) via the following procedure.

$$if\ (occupancy\_life(v_{x,y}) = 0) \quad \dots (3)$$
$$set\ robot\_id(v_{x,y}) = i;$$
$$set\ occupancy_{life(v_{x,y})} = t;$$

$$cost\left(Predecessor\left(v_{x,y}\right), v_{x,y}\right) = \infty;$$

While declaring a vertex as a dynamic obstacle it must be ensured that the vertex is unoccupied previously. Also currently the algorithm only perceives a peer-robot's path as a dynamic obstacle. In future mechanisms can be incorporated which allow us to predict the occupancy-life of an obstacle thus making algorithm more sensitive towards the spatiotemporal nature of obstacles. It must be noted in (3) that a robot ID is assigned to the robot_id variable of vertex $v_{x,y}$. This means that the vertex will be free after t milliseconds as the robot with ID *i* passes through the vertex while pursuing its path.

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 D.

3. THE PATH PLANNER

3.1 Occupancy Grid and its relation with Graph

It must be highlighted that the obstacle-map is dynamically updated by multiple cooperating robots and thus needs to be placed on a central server as a shared memory. This serves for the real-time efficiency that usual robotics navigation applications demand. Whenever an obstacle is detected, the algorithm measures the obstacle size and orientation using parameterized mean shift clustering algorithm using PCA based clustering techniques [13]. The calculated obstacle blob is placed on an occupancy grid. The resolution of the occupancy grid is kept to a suitable level so as to facilitate the placement of the robot well within a single grid location. Now the occupancy grid units that overlap with the obstacle blobs are marked as occupied. A tuning parameter *d* is attached to all obstacles which serve to dilate the size of an originally detected obstacle. This tuning parameter increases the size of the obstacle by a margin so that robots can steer with a safety margin around the obstacles without collisions. The units of occupancy grid are mapped to nodes in the eight-connected obstacle-map graph *G*, thus converting physical adjacency relationships to graph connectivity.

3.2 Algorithm

Algorithm – Multi-Robot D* Lite

procedure initialize( )
{2.1} set *robot_list = R;*
{2.2} G=initialize_obstacle_map (*S*,*D*);
//update map with start positions, initial obstacles and goal
//positions

procedure multirobot_D*Lite (*vector robot_list, graph G*)
{3.1} prioritize_robot_list (*heuristic, robot_list*);
{3.2} for all $r_i \in$ *robot_list* // for all robots in *robot_list*
{3.3} signal_adpated_D*Lite(r_i);

procedure update_obstacles(*graph G*)
{5.1} for all $v_{x,y} \in V$ where $occupancy\_life\left(v_{x,y}\right) \neq \begin{cases} 0 \\ \infty \end{cases}$
{5.2} decrement($occupany\_life\left(v_{x,y}\right)$);
{5.3} Scan for any vertices where $occupancy\_life(v_{x,y}) = 0$
{5.4} $if\ found\ \{\ set\ robot\_id(v_{x,y}) = \emptyset$;
{5.5} $cost\left(Predecessor\left(v_{x,y}\right), v_{x,y}\right) = 1;\ \}$
{5.6} Scan for any vertices where $occupancy\_life(v_{x,y}) = \infty$
{5.7} $if\ found\ \{\ cost\left(Predecessor\left(v_{x,y}\right), v_{x,y}\right) = \infty;\}$

procedure update_robot_list (*list robot_list, graph G*)
{4.1} set *robot_list*={∅};
{4.2} update_obstacles($v_{x,y}$); // as per section 2.4
{4.3} for all vertices $v_{x,y} \in V$ affected by obstacle change
{4.4} *robot_list*.add($robot\_id(v_{x,y})$);

procedure main( )
{1.1} initialize( );
{1.2} forever
{1.3} multirobot_D*Lite (*robot_list, G*);
{1.4} update_robot_list(*robot_list, G*);
{1.5} update(*S*); // update start positions to current positions
//a requirement of D*Lite

D* Lite Algorithm (Adapted for Multi-Robot D* Lite)

Complete D Lite algorithm can be referred by consulting [8]. This section only highlights the adapted part of the D*-Lite algorithm.*

Note: Each robot runs a separate instance of adapted D Lite algorithm and updates the centrally shared graph G upon completing ComputeShortestPath( ) procedure.*

procedure UpdateVertex(u)
{07'} $if\ (u \neq s_{goal})\ rhs(u) = min_{s' \in Succ(u)}\left(c(u,s') + gs';\right.$
{08'} $if\ (u \in U)\ U.Remove(u);$
{09'}
$if\ \left(\left(g(u) \neq rhs(u)\right)\ AND\ (occupancy\_life(u) = 0)\right)$
{09'} $U.insert\left(u, CalculateKey(u)\right);$

procedure Main( )
{21'} $s_{last} = s_{start}$;
⋮
{36'} $wait\ for\ the\ signal\ from\ multirobot\_D*Lite\ and$
$then\ scan\ for\ changed\ edge\ costs$
⋮
{48'} $ComputeShortestPath(\ )$;
{49'} $s = s_{start}$; $t = 1$;
{50'} $while\ (s \neq s_{goal})$
{51'} $update\_occupancy\_life(s,t)$; //set occupancy-life of
//vertex s as t
{52'} $s = argmin_{s' \in Succ(s)}(c(s,s') + g(s'))$;
{53'} $set\ robot\_id(s) = r_i$;
{54'} t=t+1000;

As first step of the algorithm, each robot initially marks obstacles over the occupancy grid on a central server (non-mobile platform). The algorithm then chooses a priority order for the robots based on a heuristic criterion. Based upon this prioritization, the server applies the adapted 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 occupancy-life (*t*) decrements to value of zero. After generation of shortest path, Adapted D*-Lite algorithm also attaches robot ID information to each vertex involved in robot path. The algorithm also attaches occupancy-life values with each vertex in robot path with a 1000 millisecond increment to each vertex as we move from start node s_i to goal node g_i along the robot path. This increment is usually the time period that robot takes to travel from one occupancy-grid location to another neighboring location.

The proposed algorithm detects the vertices for which corresponding $occupany\_life(v_{x,y})$ values either reach zero or infinity. The algorithm extracts robot ID information from such vertices and builds a priority list of robot IDs. This list is used by algorithm to selectively and sequentially run adapted D* Lite algorithm for the purpose of finding reconfigured paths due to obstacle appearance or disappearance.

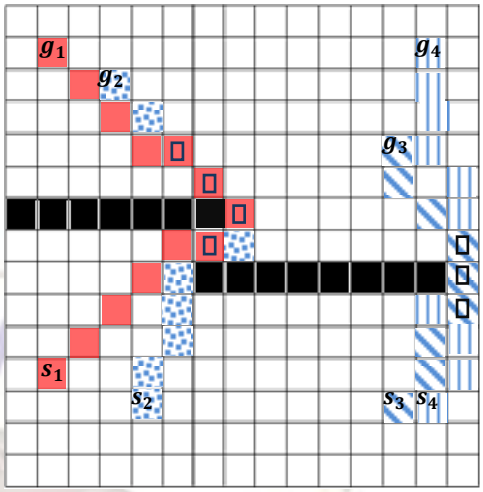

Fig. 2. Path plan by multi-robot D-Lite using minimum Euclidean distance priority heuristic*

As evident from the algorithm, all other vertices which were not disturbed by the obstacle change will retain their occupancy-time and path information.

3.3 Prioritization Heuristic

The prioritization heuristic used in step {3.1} is simply a prioritization criterion to sort the *robot_list* for sequential execution of single robot D* Lite algorithm for robots indexed in *robot_list*. Tested heuristics in our implementation include (i) number of obstacles overlapping straight line path from start to goal position. (ii) minimum Euclidean distance between start and goal position. (iii) custom-defined priority.

4. EXPERIMENTATION AND PERFORMANCE ANALYSIS

The algorithm was implemented in MRPT (mobile robot programming toolkit) for simulation purposes. A 15 x15 occupancy grid was generated for evaluating path planning capability of the algorithm. Fig. 2 and Fig. 3 depict planned paths by algorithms under different priority heuristic. The vertices containing □ symbol denote that these particular vertices were time-shared between multiple robots. Most computationally expensive steps of the algorithm were {5.3} and {5.6}. This is because virtually all vertices of the graph need to be queried in order to ascertain the value of *t* associated with each 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 $p_{cross-over}$ and probability of obstacle change $p_{obstacle-change}$. A useful relationship that can be used to quantify this factor f is

$$f \propto k.(p_{cross-over} + p_{obstacle-change})$$

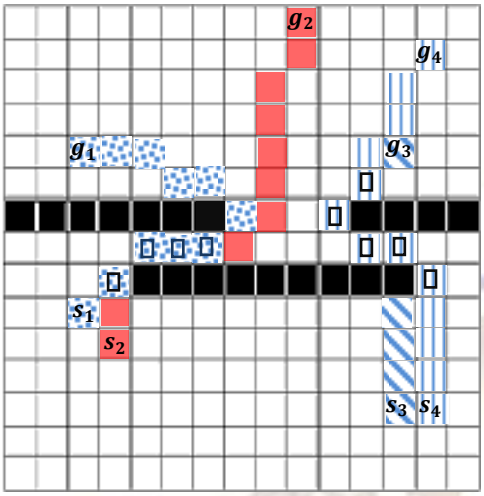

Fig. 3. Path plan by multi-robot D-Lite using min number of obstacles overlapping straight line path heuristic*

We were able to gather data for the number of node expansions, and heap-sorts (for robot priority and key sorting) performed for different priority criterion used to sequentially execute adapted D*Lite over individual robots. The data shown in Table. 1 involves a total of 225 nodes (vertices) and no obstacle changes during execution (except those caused by peer robots)

No. of robots	Priority heuristic used	Total Node Expansions	Heap Sorts
3	Minimum Euclidean distance	94	16
3	min number of obstacles overlapping straight line path	81	14
7	Minimum Euclidean distance	191	41
7	min number of obstacles overlapping straight line path	165	37
10	Minimum Euclidean distance	301	95
10	min number of obstacles overlapping straight line path	317	113

*Table. 1. Performance data: multi-robot D*Lite Algorithm*

5. CONCLUSION

In this paper, we proposed a multi-robot version of famous single robot D* Lite path-planning algorithm. The algorithm is able to maintain the real-time performance of single robot D* Lite path planning algorithm while maintaining 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. The algorithm does not optimize the velocity and sub-vertex level motion of robot to avoid collisions which has already been achieved in single robot path-planning algorithms [10]. Lastly only simulation based results were presented in the paper as the algorithm builds on top of an exhaustively implemented and tested, D* Lite algorithm.

The probabilities ($p_{obstacle-change}$ and $p_{cross-over}$) mentioned in last section need further experimental and theoretical background to be accurately quantified. An investigation into obstacle behavior (probability of changing location) and probability of path cross-over (given the dimensions of obstacles and cardinality of peer robots) will certainly lead to more efficient multi-robot path planning algorithms.

6. ACKNOWLEDGEMENT

This work is supported by NPST program by King Saud University (Project No. : 08-ELE300-02).

7. REFERENCES

[1] De Carvalho, R.N., Vidal, H.A., Vieira, P., Ribeiro, M.I., “Complete coverage path planning and guidance for cleaning robots” in *Proceedings of the IEEE International Symposium on Industrial Electronics*, 1997.

[2] E. K. Xidias, A. C. Nearchou, N. A. Aspragathos, "Vehicle scheduling in 2D shop floor environments", *Industrial Robot: An International Journal*, Vol. 36 Iss: 2, pp.176 – 183, 2009.

[3] R. Zlot, A. Stentz, “Market-based Multirobot Coordination Using Task Abstraction” in *Proceedings of 4th International Conference on Field and Service Robotics*, July 14–16, 2003.

[4] M. Takahashi, T. Suzuki, H. Shitamoto, T. Moriguchi, K. Yoshida, "Developing a mobile robot for transport applications in the hospital domain, Robotics and Autonomous Systems", Volume 58, Issue 7, *Advances in Autonomous Robots for Service and Entertainment*, 31 July 2010, Pages 889-899.

[5] B. L. Brumitt, A. Stentz, "GRAMMPS: A Generalized Mission Planner for Multiple Mobile Robots In Unstructured Environments" in *International Conference on Robotics and Automation - ICRA* , pp. 1564-1571, 1998.

[6] M. Ryan, "Exploiting Subgraph Structure in Multi-Robot Path Planning" in *Journal of Artificial Intelligence Research* Vol. 31 (2008) PP-497-542.

[7] P. Surynek, "Making Solutions of Multi-robot Path Planning Problems Shorter Using Weak Transpositions and Critical Path Parallelism" in *Proceedings of the 2009 International Symposium on Combinatorial Search*, Lake Arrowhead, CA, 2009.

[8] S. Koenig and M. Likhachev. "D* Lite" in *Proceedings of the AAAI Conference of Artificial Intelligence (AAAI)*, pages 476-483, 2002.

[9] I. Chattopadhyay, G. Mallapragada and A. Ray, "V-star: a robot path planning algorithm based on renormalised measure of probabilistic regular languages" in *International Journal of Control* Vol. 82, No. 5, May 2009, 849–867.

[10] J. Berg and M. H. Overmars, "Roadmap-Based Motion Planning in Dynamic Environments" in *IEEE Transactions on Robotics*, Vol. 21, No. 5, 2005.

[11] S. M. LaValle, "Planning Algorithms" *Cambridge University Press*, 2006.

[12] Y. Koren, J. Borenstein, "Potential Field Methods and their Inherent Limitations for Mobile Robot Navigation" in *Proceedings of the IEEE Conference on Robotics and Automation*, Sacramento, California. April, 1991. 1398-1404.

[13] T. C. Sprenger, R. Brunella, M. H. Gross, "H-BLOB: a hierarchical visual clustering method using implicit surfaces" in VIS '00 Proceedings of the conference on Visualization '00 IEEE Computer Society Press, Los Alamitos, CA, 2000.