An Overview of Optimal Graph Search Algorithms for Robot Path Planning in Dynamic or Uncertain Environments

Steven Bell *Student Member, IEEE*

IEEE Student Member No. 90691022

Oklahoma Christian University

19 March 2010

Contents

I Introduction 3

II Difficulties in Path Planning 3

III Foundations: Dijkstra's Algorithm and A* 3

IV Defining Behavior 4

V Efficiently Handling the Unknown 5
- V-A D* 5
- V-B Lifelong Planning A* 5
- V-C D* Lite 6
- V-D Additional Speed Improvements: Quadtrees 6

VI Future Advances 7

VII Conclusion 7

References 7

***Abstract*—Efficient path planning is critical to the development of useful autonomous robots, forming the glue between low-level sensory input and high-level objective completion. In this paper, we investigate two primary challenges: computing a path efficiently, and handling changes in the environment. The A* algorithm forms the foundation for solving the first problem, while D* and D* Lite solves the second problem by updating only the affected portions of the path. These incremental update methods improve the path recalculation time by as much as two orders of magnitude, enabling them to be used for real-time applications.**

I. Introduction

In simplest terms, path planning (also known as trajectory planning) is the method a robot uses to determine how to move from point A to point B [1], [2]. To generalize this definition, points are defined as multi-dimensional quantities which can represent not only position, but also orientation, configuration of a manipulator, or other variables [3]. A goal point is specified, and the planning algorithm must calculate a path from the current configuration point to the goal point. Generally, the algorithm is designed to minimize a particular factor, such as time or danger to the robot [2].

This type of planning is easy for humans: A cross-country runner can simultaneously evaluate terrain and find a near optimal route through it, even while running near top speed across rocky terrain or thick woods. For robots, however, several facets of the problem are particularly difficult. Several groups of algorithms have been developed to attack these problems, and this paper will explore the incremental development of the graph search family of algorithms. All of the primary algorithms presented in this paper are optimal; that is, they are guaranteed to always find the best path, if one exists. As will be detailed, the primary push is for increased time efficiency, so that path planning can be done in real time.

II. Difficulties in Path Planning

For many path-planning applications, the run time of the planning algorithm is not a factor. LaValle discusses several scenarios, such as using CAD models of a car factory to determine the optimal paths for assembly robot arms, where all of the trajectory planning is done as part of programming the robot. However, most mobile robots need to plan in real time, and high-dimensional problems can take hours to solve[3]. A robot is generally useless if it has to pause for an hour for every meter it drives; thus speed is one of the primary concerns in path planning. The need for computational effeciency has been the primary force pushing the development of many advances in the field.

Another complicating factor is that often the robot must operate with limited knowledge of the environment. This may be because the robot is in unexplored territory, such as an uncharted driving course or another planet like Mars. Alternatively, the robot may be working in a dynamic environment where the position of obstacles is constantly changing. This could be a restaurant, parking lot, or competition scenario such as RoboCup. In this case, the bounds of the environment are known, but cars, robots, or other obstacles are continuously moving around within it.

III. Foundations: Dijkstra's Algorithm and A*

The most popular group of path planning algorithms are based on Dijkstra's algorithm, which was developed by Edsger Dijkstra in 1959. It is a graph search algorithm; that is, its purpose is to find the shortest path through a set of interconnected nodes.

The operation of Dijkstra's algorithm in the context of 2-dimensional path planning is as follows:

1) First, the environment is overlaid with a regular grid. The algorithm will ultimately create an ordered list of points (also referred to as nodes or cells) which constitute the least-cost path through the grid.
2) Being on a point or traversing from point to point incurs a cost. In particular, a point blocked by an obstacle has infinite or prohibitively high cost. Depending on the application, the distance between points might be related to time and added to the total cost.
3) Points are iteratively assigned a score, which is the total cost to reach that point. To perform this assignment, the algorithm maintains a sorted list of nodes which need to be evaluated, termed the "OPEN list". The algorithm takes the lowest cost point from the list and calculates the score for each of its neighbors. The newly evaluated points are added to the list, and the original point is removed. This process is repeated until the goal is reached, or the OPEN list becomes empty, indicating that there is no finite-cost path to the goal. Once the goal is reached, a path can be quickly found by working backward from the goal and repeatedly moving to the cell with the greatest cost decrease. Under certain constraints, this path is guaranteed to be the least-cost path to the goal [3].

Several iterations of Dijkstra's algorithm are shown in Figure 1. It is assumed in this case that moving from

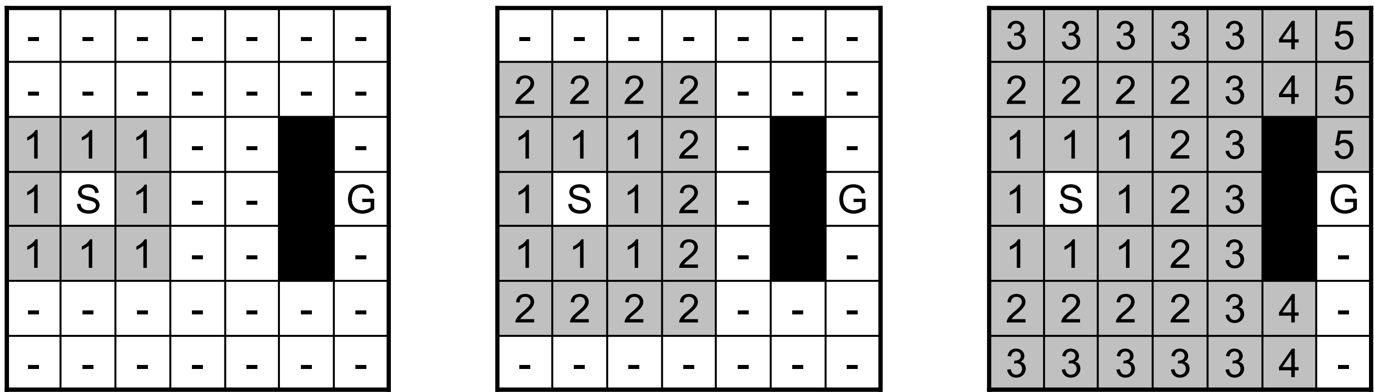

Figure 1. Two-dimensional grid operation of Dijkstra's algorithm. The start and goal positions are labeled "S" and "G" respectively. Squares whose cost has been calculated are labeled with their cost and colored gray. Impassable obstacles are black.

one square to any of the eight adjacent squares incurs a cost of one. Note that the algorithm ends up searching nearly the entire grid before reaching the goal.

First introduced in 1968 [4], A* (pronounced "A-star") became one of the foundational algorithms for modern robot path planning. It is very similar to Dijkstra's algorithm, but adds a heuristic estimate to focus the search toward the goal [3]. When picking the next point to evaluate, the algorithm also includes an estimate of how far the point is from the goal. As long as the heuristic never overestimates the distance to the goal, the algorithm will ignore low-cost options that travel away from the goal in favor of ones that move toward it.

Figure 2 shows the operation of A* relative to Dijkstra's algorithm. Far fewer nodes have been evaluated, and assuming that the time required to calculate the heuristic is small, the speed of the algorithm has been vastly improved. In MATLAB simulations with randomly-generated 10,000-point grids, A* evaluated 40% fewer nodes, but produced exactly the same path as Dijkstra's algorithm. Note that if the heuristic estimate is a constant, A* becomes identical to Dijkstra's algorithm [3], meaning that the worst-case performance of A* is the same as Dijkstra's algorithm.

IV. Defining Behavior

The most compelling strength of A* is that it is a general-purpose algorithm whose behavior can be easily modified and extended by weighting the traversal costs. "Good" traversal means different things for different robots, or can even change as a robot performs different tasks.

In the simplest navigation case, obstacles have infinite cost, so traversing through or over them is prohibited. Real life, however, is generally far more nuanced. A human runner might be faced with the option of running over rough ground, wading through a swamp, or taking the long way around. There is no good choice, and it might just happen that wading through the swamp is the fastest route to the goal. The Mars Exploration Rovers (MERs) use an algorithm its developers term T*, which analyzes the terrain around it and classifies the terrain into areas of high, moderate, low, and poor traversability [5]. Where black-and-white A* attempts to drive only on perfectly safe terrain, T* weights the traversal cost based on the terrain rating. This means that the rover may decide to take the short path over "moderately traversable" terrain, rather than take the long route on "highly traversable" terrain. In cases where the goal itself lies in an area of less-than-optimal traversability, A* will fail to find a path, but T* still works [5]. An updated algorithm developed since the landing of the MERs uses the same process, but incorporates analysis and weighting of other hazardous terrain, such as sand dunes or slippery inclines [6].

In the RoboCup competition, the field is smooth and flat. In this environment, the "goodness" of a path has nothing to do with traversability. Instead, the cost can be related to the time to travel the path and the likelihood of colliding with an opponent [7]. By estimating the velocity of opposing robots and penalizing paths that intersect, robots take paths that are more likely to be unobstructed.

Other behaviors are easy to imagine. Stentz discusses the possibility of a stealthy robot [8], which favors paths that follow the perimeter of objects, and avoids long drives through the open. This behavior could be produced by simply increasing the traversal cost of large open areas or decreasing the cost of driving near an obstacle. Likewise, a solar-powered robot could easily be designed to prefer routes in the sun, or a military robot made to choose known terrain over unknown areas.

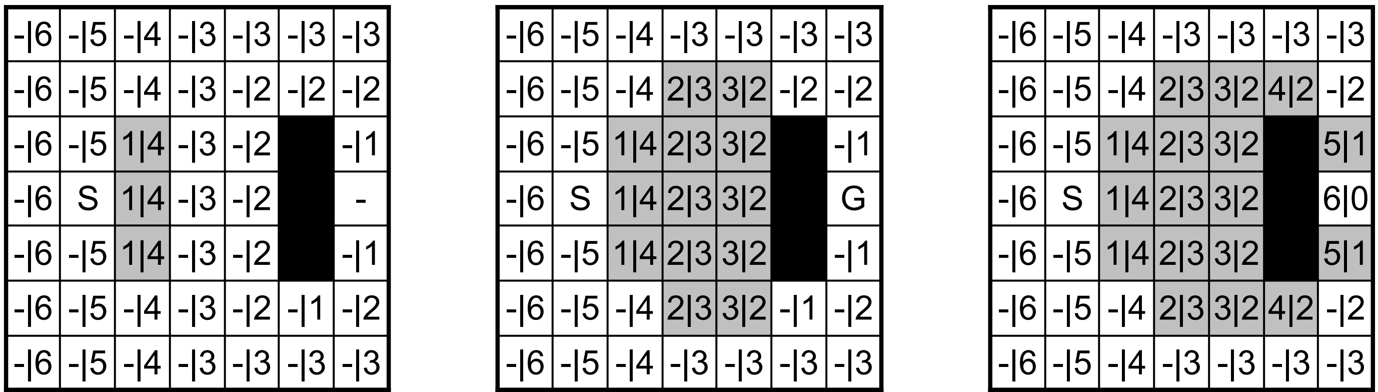

Figure 2. Two-dimensional grid operation of A*. The number on the right half of each cell is the estimated cost, the left half contains the score. Note that the estimated cost would not actually be calculated for every cell - the heuristic is only calculated for cells on the OPEN list.

V. Efficiently Handling the Unknown

The basic theory of driving in an unknown environment is straightforward: the robot builds a path based on what it knows, and follows that path until it encounters an obstacle. It then adds the new obstacle to its map, and plans a new path from its current location to the goal. The time spent recalculating is hardly a concern in our theoretical two-dimensional worlds where we only occasionally add new obstacles for the algorithm to handle. However, robots in the real world must often work with massive, high resolution grids. Additionally, they may get new map data with every sensor frame, particularly if they utilize a laser scanner or other ranging device. In this scenario, the percentage of the time the robot spends recalculating paths is suddenly staggering. The problem is that A* has no memory of past paths and no way to adjust its current path to handle new obstacles.

*A. D**

D*, which derives its name from “Dynamic A*” [2] modifies A* such that traversal costs can be dynamically increased and decreased. When the robot finds an obstacle, the algorithm marks the neighboring nodes as RAISED; that is, the cost to reach them has increased. The algorithm checks to see if each RAISED node can be given a lower score based on a different neighbor. If so, it is flagged as LOWERED. If not, the RAISE state is passed to each of the node’s neighbors. These waves of RAISE and LOWER states propagate through the grid until they reach the robot, indicating that a new optimal path has been found. Stentz shows experimental results where D* outperforms A* by two orders of magnitude for a 1 million point grid [2], fast enough to run in real time.

Focussed D* is an extension to D* [9], which uses a heuristic to focus the waves toward the robot in the same way that A* focuses the search toward the goal. In simulations, Focussed D* has been shown to be up to two times faster than D* for large gridworlds [9]. However, newer and simpler algorithms have mostly supplanted Focussed D* in recent years.

*B. Lifelong Planning A**

Lifelong Planning A* (LPA*), as its name suggests, is designed to speed up repeated replanning by saving information from earlier in the algorithm’s “life”. LPA* was developed as a combination of two algorithms: A*, which speeds up static path planning as described above, and DynamicSWSF-FP, which speeds up replanning. The latter is a variation of Dijkstra’s algorithm designed to dynamically handle changes to traversal costs by reusing unaffected portions of the path [10].

LPA* behaves exactly like A* during its initial run. However, once a traversal cost changes due to a new obstacle or other cause, LPA* begins updating nodes rather than re-running the algorithm from scratch. The key to doing this effeciently is to update only the nodes that are both affected by the cost change and are essential to finding the new path.

In order to do this, LPA* maintains two scores for each node: the current score, and the lowest neighbor’s score plus the traversal cost. If these scores are equal, then the node is “locally consistent”; if not, it needs to be evaluated. These locally inconsistent nodes are marked with unknown cost and added back into the OPEN list. From here, the algorithm can proceed essentially as before, evaluating nodes and moving toward the goal. In this way, changed nodes which are not part of the path calculation are ignored, which saves computation time.

Figures 3 and 4 illustrate the operation of LPA*. In figure 3, the initial state is shown, where the cost to reach

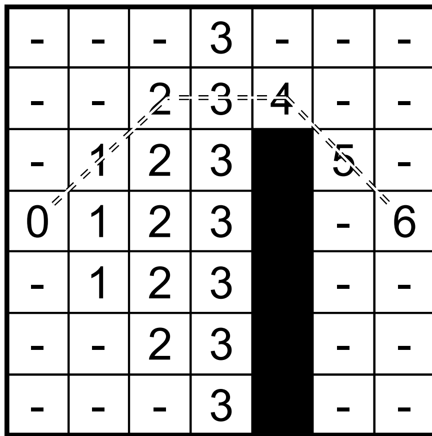

Figure 3. Initial path traced with A*. As in previous examples, the traversal cost between cells is always 1; black obstacles have infinite cost.

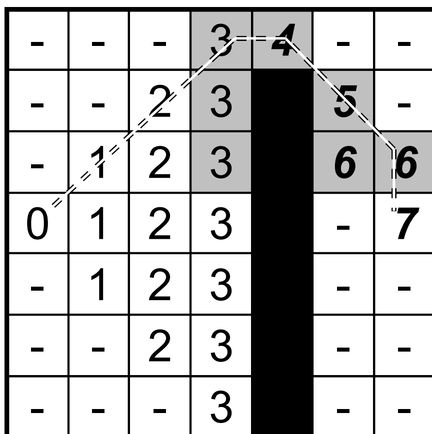

Figure 4. Replanned path using LPA*. Only the gray cells had to be checked, and only the italicized cells had their scores changed.

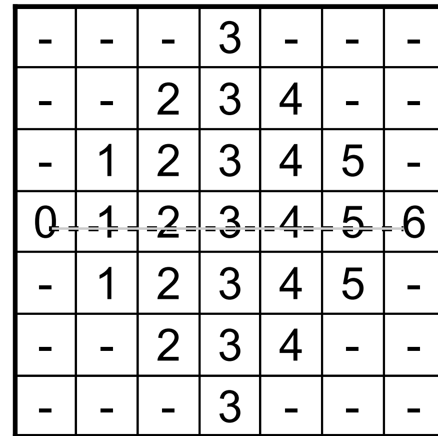

Figure 5. Initial path calculated with LPA* in an empty field. Dashes represent uncalculated values.

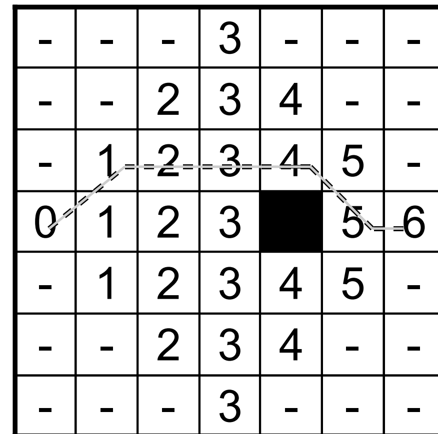

Figure 6. Replanned path, which does not require any changes to node scores.

each node is marked. In figure 4, the obstacle is found to be larger than originally mapped. The surrounding nodes, marked in gray, must be re-checked. However, the unmarked cells near the origin do not need to be checked.

In sparse environments, it is even possible that no nodes need to be updated. Figures 5 and 6 illustrate this possibility. Because the points on the right side of the obstacle can be reached without any additional cost, the new path is no more costly than the original. Were A* being used to update the node scores, every one of the numbered cells would have to be recalculated.

C. D Lite*

D* Lite builds on LPA* to create an algorithm similar to D* [11]. Like D*, it reverses the direction of search, working from the goal toward the robot. This way, the computed scores will always represent the cost to the goal, rather than the cost from the starting location. Additionally, D* Lite tweaks the way the heuristics are calculated, so that the priority queue holding the OPEN list requires fewer resorting operations. Ultimately, D* Lite performs the same function as the original and Focussed D*, but requires fewer lines of code and is simpler to understand. Performance-wise, it is not “lite” at all - it runs as fast or faster than D* [11].

D. Additional Speed Improvements: Quadtrees

As mentioned before, a prerequisite to solving for a path is to partition the environment into grid squares. Not surprisingly, the search time is a function of the number of points in the grid. For applications which have large environments or require high precision, the resolution of the grid can become a prohibitive factor.

A quadtree is a data structure consisting of a tree of nodes where each node has up to four children. For terrain representation, each node either holds information about that square or has pointers to four children which each cover one quadrant [1]. In this way, a

variable-density rectangular mesh is created, allowing high resolution representation near the robot and lower resolutions farther away. Yahja et al. note that "Natural terrains are usually sparsely populated and are often not completely known in advance." They go on to show that for environments with low obstacle density, a quadtree representation can vastly speed up path computation [1].

VI. Future Advances

Many open questions and challenges remain in the relatively new field of robotic path planning. Speed will continue to be a dominating factor in the development of improved algorithms. Despite sporting impressive computing power, most robots competing in the DARPA Urban Challenge stayed in the range of 5-20 miles per hour [12]. To be useful as a military or commercial vehicle, a robot must be able to operate at normal traffic speeds. Similarly, the current generation of Mars rovers move only a couple hundred meters on a good sol (martian day). If future rovers' autonomy can be improved, they will be able to drive farther and explore more terrain in their limited operating life.

The A* family of algorithms will be extended and applied to a myriad of different behaviors beyond merely avoiding obstacles, such as those outlined in section IV. Because of the flexible nature of these algorithms, they will most likely continue to hold the field for the near future. Several entirely different classes of algorithms are being developed, including genetic algorithms [13] and rapidly–exploring random tree (RRT) algorithms [14]. In contrast to the A* family, which analyzes a large set of nodes to find the optimal path, these algorithms attempt to find a "good enough" path with a fraction of the computational effort. In the case of a genetic algorithm, a group of paths are randomly planned, and the best are chosen for reproduction and controlled mutation. By repeating this process dozens of times, a near-optimal path emerges. This approach can produce near-optimal answers for problems which are otherwise computationally intractable. Similarly, RRT algorithms recursively select a set of random sub-paths to check, rather than planning node by node. These algorithms based on random "guess-and-check" may ultimately dominate, particularly for applications which must handle several dimensions with very limited computational resources.

VII. Conclusion

This paper has presented the leading algorithms for autonomous robot path planning, particularly A*, LPA*, and D* Lite. Each of these methods represents an incremental improvement in computational efficiency, which has ultimately enabled the use of graph search path-planning algorithms in complex real world environments. By using optimizations to speed up path recalculation, these algorithms have become viable for many robotic applications, including planetary rovers and autonomous cars. As the field of robotics pushes forward, path planning will continue to play a major role in the behavior and intelligence of robots - and for now, optimal graph search algorithms will lead the pack.

Originality Statement

All of the algorithms presented were developed by others. However, I implemented Dijkstra's algorithm and A* in MATLAB, which bolstered my understanding and description of these algorithms.

Acknowledgments

I would like to thank Dr. David Waldo for his careful review and criticism of earlier drafts, and for encouraging me to pursue submission of this paper. I am also grateful to Professor Joe Watson and the OKC IEEE section for their support and comments on both the paper and presentation.

I was the sole author of this paper, and wrote all of the supporting MATLAB code mentioned above.

References

[1] A. Yahja, A. Stentz, and S. Singh, "Framed-Quadtree Path Planning for Mobile Robots Operating in Sparse Environments," in *IEEE Conference on Robotics and Automation*, (Leuven, Belgium), pp. 650–655, May 1998.

[2] A. Stentz, "Optimal and Efficient Path Planning for Partially-Known Environments," in *IEEE Conference on Robotics and Automation*, vol. 10, (San Diego, CA), pp. 3310–3317, 1994.

[3] S. LaValle, *Planning Algorithms*. Cambridge, UK: Cambridge University Press, 2006.

[4] P. Hart, N. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[5] A. Howard, H. Seraji, and B. Werger, "T*: A Novel Terrain-Based Path Planning Method for Mobile Robots," tech. rep., NASA Jet Propulsion Laboratory, 2002.

[6] D. Helmick, A. Angelova, and L. Matthies, "Terrain Adaptive Navigation for Planetary Rovers," *Journal of Field Robotics*, vol. 26, no. 4, pp. 391–410, 2009.

[7] A. Farinelli and L. Iocchi, "Planning Trajectories in Dynamic Environments Using a Gradient Method," in *RoboCup 2003: Robot Soccer World Cup VII* (D. Polani, ed.), (Berlin, Germany), Springer, 2004.

[8] A. Stentz, "Constrained Dynamic Route Planning for Unmanned Ground Vehicles," in *Proceedings of the 23rd Army Science Conference*, (Orlando, FL), December 2002.

[9] A. Stentz, "The Focussed D* Algorithm for Real-Time Replanning," in *International Joint Conference on Artificial Intelligence*, vol. 14, pp. 1652–1659, 1995.

[10] G. Ramalingam and T. Reps, "An Incremental Algorithm for a Generalization of the Shortest-Path Problem," *Journal of Algorithms*, vol. 21, no. 2, pp. 267–305, 1996.

[11] S. Koenig and M. Likhachev, "D* Lite," in *Proceedings on the National Conference on Artificial Intelligence*, (Menlo Park, CA), pp. 476–483, AAAI Press, 2002.

[12] Y.-L. Chen, V. Sundareswaran, and C. Anderson, "TerraMax: Team Oshkosh Urban Robot," *Journal of Field Robotics*, vol. 25, no. 10, pp. 841–860, 2008.

[13] K. Sugihara and J. Smith, "A Genetic Algorithm for 3-D Path Planning of a Mobile Robot," 1996.

[14] N. A. Melchior and R. Simmons, "Particle RRT for Path Planning with Uncertainty," in *2007 IEEE Conference on Robotics and Automation*, (Roma), pp. 1617–1624, 2007.