Efficient Optimal Search of Euclidean-Cost Grids and Lattices

James J. Kuffner

The Robotics Institute
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213, USA
email: kuffner@cs.cmu.edu

Digital Human Research Center
National Institute of Advanced
Science and Technology (AIST)
2-41-6 Aomi, Koto-ku, Tokyo, Japan 135-0064

***Abstract*— We describe a simple technique to speed up optimal path planning on Euclidean-cost grids and lattices. Many robot navigation planning algorithms build approximate grid representations of the environment and use Djikstra's algorithm or A* to search the resulting embedded graph for an optimal path between given start and goal locations. However, the classical implementations of these search algorithms were designed to find optimal paths on arbitrary graphs with edges having arbitrary positive weight values.**

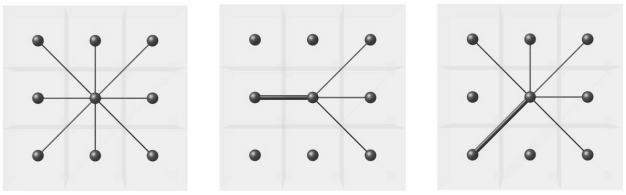
This paper explains how to exploit the structure of optimal paths on Euclidean-cost grids and lattices in order to reduce the number of neighboring nodes considered during a node expansion step. The result is a reduction in both the total nodes examined as well as the overall cost of the search. The algorithm presented increases the efficiency of robot navigation planning on 2D and 3D grids, and generalizes to any other search problem that involves searching Euclidean-cost grids and lattices in higher dimensions.

I. Introduction

Classical grid search is a well-known topic in robotics and artificial intelligence research, and has strong connections to research in dynamic programming, optimization, and algorithms for computer networks. Because the storage and computational costs for grids generally grows exponentially according to size and dimension of the grid, their use has generally been limited to low dimensional problems, particularly in robot path planning. However, grids in higher dimensions have recently been reconsidered as a deterministic alternative to path planning based on random sampling [1].

We describe a simple technique to speed up optimal path planning on Euclidean-cost grids and lattices. This technique does not improve the fundamental exponential growth of the cost of grid search, but rather improves the "constant factor" in the running time (see Section V). As an example, many robot navigation planning algorithms build approximate grid representations of the environment and use Djikstra's algorithm or A* to search the resulting embedded graph for an optimal path between given start and goal locations. The classical implementations of these search algorithms were designed to find optimal paths on arbitrary graphs with edges having arbitrary positive weight values. By contrast, embedded graphs on Euclidean-cost grids have a fixed number of neighboring nodes and known edge weights. This information can be utilized to improve the overall efficiency of a number of computations related to path planning between grid locations.

We previously derived an improved search method for the special case of navigation path planning on a 2D grid [2]. This paper generalizes that result and explains how to exploit the structure of optimal paths on Euclidean-cost grids and lattices in order to reduce the number of neighboring nodes considered during a node expansion step, the key operation in graph search algorithms. The result is a reduction in the total nodes that must be examined and ultimately stored in the priority queue. This produces a moderate to significant reduction in the overall cost of the search depending upon the difficulty of the planning query. The savings is particularly noticeable for grids and lattices with edges between diagonally-adjacent grid points that are assigned a relative cost based on the approximate Euclidean distance. We demonstrate computed examples that show significantly improved computational efficiency for robot navigation planning on 2D and 3D grids, as well as on Euclidean-cost grids and lattices in higher dimensions. However, the benefit of the optimization diminishes as the dimension d increases, and becomes negligible at around $d > 12$.



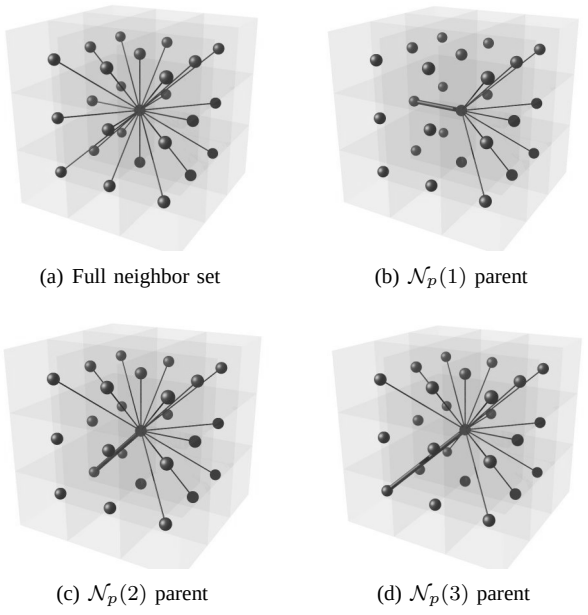
(a) Full neighbor set (b) $\mathcal{N}_p(1)$ parent (c) $\mathcal{N}_p(2)$ parent

Fig. 1. Reducing neighbor expansions on 2D grids

The rest of the paper is organized as follows: Section II gives an overview of related research, Section III discusses issues related to path search on grids, Section IV presents techniques to optimize search on Euclidean grids, Section V contains analysis of the optimized performance, Section VI presents experimental results, and Section VII concludes with a summary discussion.

II. Background

The theory and analysis of path planning algorithms is fairly well-developed in the robotics literature, and is not discussed in detail here. For a broad background in motion planning, readers are referred to [3]. For any path planning technique, it is important to minimize the number of degrees of freedom (DOFs), since the time complexity of known algorithms grows exponentially with the dimension of the $\mathcal{C}$-space [4]. A *complete* planning algorithm will



(a) Full neighbor set (b) $\mathcal{N}_p(1)$ parent

(c) $\mathcal{N}_p(2)$ parent (d) $\mathcal{N}_p(3)$ parent

Fig. 2. Reducing neighbor expansions on 3D grids

in finite time always find a path if one exists, and report failure (also in finite time) if no path exists. Complete algebraic solutions exist for the general path planning problem [5], [6], but have yet to be implemented. For problems in low-dimensional configuration spaces in which the shape of $\mathcal{C}$-obstacles is known, complete or resolution-complete methods have been devised, such as exact-cell decomposition, visibility graphs, or approximate cell-decomposition (see [3]). A popular technique for mobile robot navigation consists of discretizing the environment into a regular 2D grid and marking cells which are obstructed by obstacles. The navigation planning problem can then be cast as a search problem on the embedded graph defined by the grid. This is one instance of an approximate cell-decomposition planning method, and by adjusting the fineness of the discretization of the grid, it is possible to devise resolution-complete planning algorithms.

A. Searching for Shortest Paths on Graphs

The problem of calculating shortest paths on a weighted graph arises very often in Computer Science. A number of classical graph search algorithms have been developed, with two popular ones being Dijkstra's algorithm [7], and A* search [8]. Both algorithms compute the optmal path, and can be considered as special forms of dynamic programming [9]. A* operates essentially the same as Dijkstra's algorithm except that a heuristic function that estimates the cost to the goal is added to the cost of a node when inserting it into the priority queue. This causes the algorithm to expand more promising nodes first, potentially saving a significant amount of computation. For a discussion of the optimality of A*, see [10]. Using the heuristic function alone results in best-first search (BFS), which is a greedy algorithm that can sometimes vastly reduce computation times compared to Dijkstra's algorithm or A*. However, path optimality is no longer guaranteed. Large graphs typically benefit greatly from the use of A* or BFS along with a reasonable heuristic function. There also exists a linear time algorithm due to Henzinger, Klein, and Rao for computing all shortest paths from a single source in large planar graphs [11]. However, due to the overhead involved in running the algorithm, the potential execution time savings may only be realized for very large graphs. Other search strategies of interest cache information for dynamic or unknown environments in order to avoid having to search from scratch each time, such as the D* (Dynamic A*) algorithm [12].

III. Planning on Grids and Lattices

Grids and lattices implicitly define a connected graph, so path planning between two grid points can be simply reduced to graph search. However, the classical implementations of these search algorithms were designed to find optimal paths on arbitrary graphs with edges having arbitrary positive weight values. By contrast, embedded graphs have a fixed number of neighboring nodes. In addition, in the case of a Euclidean-cost grids, all edge weights are of known values. This information can be exploited to improve the efficiency of planning on grids, specifically for cases in which edges between diagonally-adjacent grid points are present.

A. Mathematical Formulation

Although our optimizations are most effective for grids of two or three dimensions, we will adopt notation that generalizes to grids (lattices) of arbitrary dimension. Let p be a point on a grid $\mathcal{G}$ of dimension d. Specifically, p is a vector of size d of integer values which range from the minimal and maximal extents of the grid along each axis:

$$p = (a_1, a_2, \ldots, a_d) \;\; a_i \in \mathcal{A}_i$$

Each $\mathcal{A}_i$ is the set of all integers on the range $[min_i, \ldots, max_i]$. For simplicity, let us assume that the range of integer grid component values along each exis is uniform and given by the set $\mathcal{A}_i = 1, 2, \ldots, N, i \in 1, 2, \ldots, d$. Thus, a grid $\mathcal{G}$ of dimension d will have N^d distinct grid points $p \in \mathcal{A}_1 \times \mathcal{A}_2 \times \ldots \times \mathcal{A}_d$.

Each grid point p has a fixed set of neighboring points. For the moment, let us ignore the special case of points on the grid boundary, and focus on interior grid points. Each interior point p has a set of neighboring points whose grid coordinates differ only by -1, 0, or $+1$ for all components. Let us define the *neighbors* of p, as the set $\mathcal{N}_p$ of all grid points that have all component values differing only by $[-1, 0, +1]$. Each interior point has exactly $3^d - 1$ neighbors distinct from itself. The number of grid component values that differ from p is one measure of the "closeness" of that neighbor. At this point, we will make a special distinction between *straight* neighbors and *diagonal* neighbors. Let us define the *straight* neighbors of p as the set $\mathcal{S}_p \subset \mathcal{N}_p$ of neighboring points whose grid coordinates differ only by a single component. There are exactly $2d$ straight neighbors for each interior grid point, namely two for each component

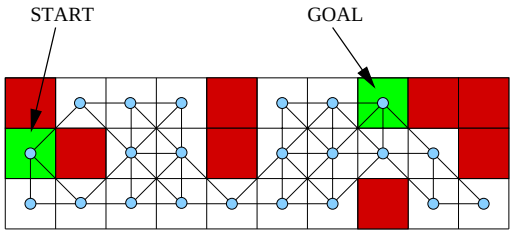


Fig. 3. Searching an embedded graph defined by a grid

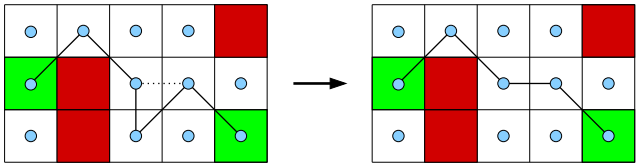


Fig. 4. Paths on 2D grids which form angles less than 90 degrees can always be shortened.

axis. All other neighbors will be referred to as *diagonal* neighbors, and make up the set $\mathcal{D}_p = \mathcal{N}_p - \mathcal{S}_p$.

More generally, we can partition the set $\mathcal{N}_p$ according to the number of component values that differ from p. Those that differ by a single value form the set $\mathcal{N}_p(1)$, which is just the straight neighbors $\mathcal{S}_p$. Similarly, we define the sets $\mathcal{N}_p(2)$, $\mathcal{N}_p(3)$, up to $\mathcal{N}_p(d)$. These sets are analagous to the set partitions defined by the *hamming distance* measure, which is used to compare strings of binary digits. Note that the special case $\mathcal{N}_p(0)$ is the point p itself, and is usually implicitly omitted from the set $\mathcal{N}_p$.

B. Robot Navigation Planning

In the case of robot navigation planning on a 2D grid, cells in an occupancy grid are marked as either *FREE* or *NOT-FREE*, depending upon whether or not the corresponding location in the environment represents a valid location for the robot. Each cell in the grid corresponds to a node in the embedded graph, and edges are placed between pairs of neighboring pixels that are both *FREE*. By searching the graph we can conservatively determine whether or not a collision-free path exists from the start location to the goal location at the current grid resolution. Considering only straight neighbors limits us to searching motions along the cardinal directions N, S, E, W. This is sometimes called 4-neighbor search. However, in order to produce shorter paths, usually diagonal motions are also considered (8-neighbor search). Figure 3 shows a small example of an embedded graph defined by a 2D grid. By assigning a relative cost to each edge we can search for a path that connects the start and goal while minimizing a cost function defined by the edge weights.

C. Assigning Edge Weights

In the case of Euclidean-cost grids, all edge weights are of known value. As the name implies, Euclidean-cost grids utilize the relative Euclidean distance between adjacent grid points in assigning edge weights. All edges between straight neighbors are assigned a relative cost of 1, while edges between diagonal neighbors are assigned a relative cost based on the Euclidean distance. For example, diagonal neighbors on a 2D grid are assigned a relative weight of $\sqrt{2} \approx 1.4$. For the 3D case, $\mathcal{N}_p(2)$ and $\mathcal{N}_p(3)$ diagonal neighbors are assigned weights of $\sqrt{2}$ and $\sqrt{3}$ respectively. For the general case, edges connecting a grid point p to all neighboring points $q \in \mathcal{N}_p(k)$ are assigned a relative weight:

$$||p - q||_2 = \sqrt{k} : \; q \in \mathcal{N}_p(k) \; , \; k \in [1, 2, \ldots, d]$$

The nature of these fixed costs can be utilized to improve the overall efficiency of a number of computations related to path planning between grid locations.

IV. Optimizing Grid Search

In this section, we show how to speed up path searching on grids with straight and diagonal edge weights assigned as described in the previous section. The basic idea is to modify A* and Dijkstra's algorithm so as to reduce the overall total number of expected neighboring node expansions by limiting the number of neighboring cells inserted into the priority queue. We can do so by exploiting the geometry of optimal paths on a grid with Euclidean edge weights.

For example, consider the case of an optimal 8-neighbor path computed on a 2D grid. The minimum angle an optimal path on a 2D grid can ever form with itself is 90 degrees. This fact is apparent by noting that any 8-neighbor path that forms an angle smaller than 90 degrees can be made shorter by "cutting corners" (see Figure 4). Because of this, we can effectively limit the number of neighboring node expansions by ignoring those cells whose inclusion would produce a path with an angle less than 90 degrees. By keeping track of which direction a given node was expanded from, we can reduce the number of potential node expansions from 8 to 3 for $\mathcal{N}_p(1)$ neighbors and from 8 to 5 for $\mathcal{N}_p(2)$ neighbors. This is graphically illustrated in Figure 1. For example, suppose we extract a node from the queue that was inserted when its neighbor node to the "west" was expanded (Figure 1(b)). Obviously, there is no need to consider the node to the west when the current node is expanded. In addition, there is no need to consider the neighbor nodes to the northwest and southwest. This is because the edges connecting them to the current node *cannot* be part of an optimal path that includes the edge between the current node and its western neighbor.

A similar argument can be made for the case of paths on 3D grids with Euclidean weights. Figure 2 graphically illustrates the different cases that arise depending upon the relative location of the *parent* point. The parent point is the grid location that corresponds to the neighboring node from which the current node was originally expanded from. In general, when a node corresponding to a grid point p is expanded, its parent point q along with all of the the mutual neighbors of q can be excluded. To see why, let r be a mutual neighbor of both p and q:

$$r \in \mathcal{N}_p \cap \mathcal{N}_q$$

The edge formed between q and p, and p and r cannot simultaneously be a member of an optimal path, because the edge between q and r could be used instead, and will always be of lower cost due to the triangle inequality.

There are several properties of optimal paths on grids with Euclidean weights that are useful to note: *Multiplicity*: An optimal path will generally not be unique, as there may be several optimal paths of equivalent length. A complete optimal planner is guaranteed to return one of them. *No Crossings*: Optimal paths will never cross themselves, otherwise any path loops could simply be eliminated to yield a shorter path. *Bounded Curvature*: Optimal paths have a bounded curvature due to the Euclidean edge wieghts (e.g. the minimum angle an optimal path on a 2D grid can ever form with itself is 90 degrees). In the next section, we compare the sizes of the neighbor node expansion sets for both traditional and optimized grid planning.

V. Analysis

We first compare the relative number of node expansions for grids of different dimensions. For traditional search which uses the full set of neighbors, as illustrated in Figure 1(a) and Figure 2(a). For a grid of dimension d, the total number of neighbors of a point p (i.e. the complete set $\mathcal{N}_p$) is given by $3^d - 1$. The sizes of the sets partitioned according to the number of component values that differ from p can be expressed as a recurrence relation. Let $\mathcal{N}_p^i(k)$ be the set of all neighbors with k component values that differ from p on a grid of dimension i. The size of this set is defined recursively as:

$$|\mathcal{N}_p^i(k)| = \begin{cases} 1 & : \quad k = 0\,, \quad 1 \leq i \leq d \\ 2^i & : \quad k = i\,, \quad 1 \leq i \leq d \\ |\mathcal{N}_p^{i-1}(k)| + 2|\mathcal{N}_p^{i-1}(k-1)| & \\ \qquad & : \quad 1 \leq k < i \leq d \end{cases} \tag{1}$$

This recurrence has two base cases: 1) if $k = 0$, the set size is 1, namely it contains only the point p itself; 2) if $k = i$, then the set size is 2^i, namely the set of all maximally distant "corner" neighbors. For all other values of i and k, subject to the condition $1 \leq k < i \leq d$, the size of the set is given by the sum of of the recurrence in Equation 1.

Table I shows the computed set size values for grids of different dimensions. We verify that the sum of all of the partition set sizes for a grid dimension d matches the total number of neighbor nodes:

$$\begin{aligned} |\mathcal{N}_p(d)| &= \sum_{k=1}^{d} |\mathcal{N}_p^d(k)| \\ &= 3^d - 1 \end{aligned}$$

We now derive the size of the optimized set of neighbor nodes $|\mathcal{M}_{p,q}|$, which omits neighboring nodes of the grid point p that can be safely excluded from the node expansion step depending upon the relative direction of the parent point q. Recall that when a node is expanded, its parent point q along with all of the the mutual neighbors of q can be excluded:

$$|\mathcal{M}_{p,q}| = |\mathcal{N}_p| - |\mathcal{N}_p \cap \mathcal{N}_q| - 1 \tag{2}$$

The size of the set $\mathcal{M}_{p,q}$ is dependent upon which neighbor partition set of p that q belongs to. As the number of differing grid component values between points p and q decreases, there will be a larger overlap in their full neighbor sets $\mathcal{N}_p$ and $\mathcal{N}_q$. This results in a correspondingly smaller size for the reduced neighbor set $\mathcal{M}_{p,q}$. This can also be expressed as a recurrence relation. Let $\mathcal{O}_p^i(k)$ be the reduced set of all neighbors with k component values that differ from p on a grid of dimension i. The size of this set is defined recursively as:

$$|\mathcal{O}_p^i(k)| = \begin{cases} 3^k - 2^k & : \quad k = i\,, \quad 1 \leq i \leq d \\ 3|\mathcal{O}_p^{i-1}(k)| & : \quad 1 \leq k < i \leq d \end{cases} \tag{3}$$

This base case occurs if $k = i$, in which case the size of the set is $3^k - 2^k$. For all other values of i and k, subject to the condition $1 \leq k < i \leq d$, the size of the reduced set is given by the sum of of the recurrence in Equation 3. Table II shows the computed total reduced set size values $|\mathcal{M}_{p,q}|$ for grids of different dimensions, and for different parent point q classifications. The rightmost column shows a calculated weighted average of the total expected number of neighbor expansions for a grid of d dimensions. These data show that the largest performance savings occur for smaller values of i and k. We have performed a number of tests in order to confirm this performance advantage.

TABLE I
Total node expansions for traditional search

Dim.	$\mathcal{N}_p(1)$	$\mathcal{N}_p(2)$	$\mathcal{N}_p(3)$	$\mathcal{N}_p(4)$	$\mathcal{N}_p(5)$	all $\mathcal{N}_p$
2	4	4				8
3	6	12	8			26
4	8	24	32	16		80
5	10	40	80	80	32	242
6	12	60	160	240	192	728
7	14	84	280	560	672	2186
8	16	112	448	1120	1792	6560
9	18	144	672	2016	4032	19682
10	20	180	960	3360	8064	59048
⋮	⋮	⋮	⋮	⋮	⋮	⋮
d	$2d$	…	…	…	…	$3^d - 1$

VI. Results

We now describe some experimental results. We first implemented a simple test program to evaluate the performance advantages when using the reduced neighbor set when path planning on multi-dimensional grids. Table III shows the computed relative frequencies of the different parent point q classifications. The data show that from this distribution, as the gird dimension d increases, the sizes of the sets of diagonal edges increases at a very rapid rate and ultimately form the bulk of the distribution.

TABLE II

TOTAL REDUCED SET SIZE BASED ON THE CLASSIFICATION OF q.

Dim.	$\mathcal{N}_p(1)$	$\mathcal{N}_p(2)$	$\mathcal{N}_p(3)$	$\mathcal{N}_p(4)$	$\mathcal{N}_p(5)$	Avg. $\mathcal{N}_p$
2	3	5				4
3	9	15	19			15
4	27	45	57	65		52
5	81	1 35	171	195	211	175
6	243	405	513	585	633	568
7	729	1215	1539	1755	1899	1811
8	2187	3645	4617	5265	5697	5683
9	6561	10935	13851	15795	17091	17634
10	19683	32805	41553	47385	51273	54266

TABLE III

APPROXIMATE FREQUENCIES ACCORDING TO q CLASSIFICATION.

Dim.	$\mathcal{N}_p(1)$	$\mathcal{N}_p(2)$	$\mathcal{N}_p(3)$	$\mathcal{N}_p(4)$	$\mathcal{N}_p(5)$
2	0.5000	0.5000			
3	0.2308	0.4615	0.3077		
4	0.1000	0.3000	0.4000	0.2000	
5	0.0413	0.1653	0.3306	0.3306	0.1322
6	0.0165	0.0824	0.2198	0.3297	0.2637
7	0.0064	0.0384	0.1281	0.2562	0.3074
8	0.0024	0.0171	0.0683	0.1707	0.2732
9	0.0009	0.0073	0.0341	0.1024	0.2049
10	0.0003	0.0030	0.0163	0.0569	0.1366

Based on the frequency distribution in Table III, we calculated a weighted average of the total expected number of neighbor expansions for a grid of d dimensions, and computed the *performance ratio* relative to full-neighbor expansion:

$$r_{perf} = \frac{\sum_{k=1}^{d} w_k \mathcal{M}_{p,q}}{\mathcal{N}_p}$$

The results are displayed in Table IV. For 2D and 3D grids, using the reduced neighbor set yields a %50 and %43 savings in computation related to node expansion. The performance advantage tapers off dramatically as the dimension increases, but still manages to maintain a %22 advantage for six-dimensional grids, and an approximately %9 savings for ten-dimensional grids.

Note that these are the ideal performance ratios, and the actual performance gains may be larger or smaller depending upon a number of interrelated factors. Parameters affecting the performance include the dimension and size of the grid, the obstacle arrangements relative to the start and goal positions, and the efficiency of the implementation data structures such as the priority queue used to manage node expansion.

In addition to the basic implementation for testing performance on grids of higher dimensions, we have also developed an interactive navigation path planning application. We implemented the optimized 2D grid search described in this paper have integrated it into a graphical user interface on a standard desktop PC running Linux. Interactive performance is observed, even for relatively large and complex environments. Figure 5 and Figure 6 show several snapshots of an interactive session involving a humanoid figure navigating in both a maze and an office environment. The user can dynamically reposition obstacles and the goal location as the character moves. The planner rapidly computes a new path based changes in the environment.

TABLE IV

PERFORMANCE RATIOS FOR THE REDUCED VS. FULL NEIGHBOR SET.

Dim.	$\mathcal{N}_p(1)$	$\mathcal{N}_p(2)$	$\mathcal{N}_p(3)$	$\mathcal{N}_p(4)$	$\mathcal{N}_p(5)$	Avg. Ratio
2	0.3750	0.6250				0.5000
3	0.3462	0.5769	0.7308			0.5710
4	0.3375	0.5625	0.7125	0.8125		0.6500
5	0.3347	0.5579	0.7066	0.8058	0.8719	0.7213
6	0.3338	0.5563	0.7047	0.8036	0.8695	0.7808
7	0.3335	0.5558	0.7040	0.8028	0.8687	0.8286
8	0.3334	0.5556	0.7038	0.8026	0.8684	0.8663
9	0.3334	0.5556	0.7037	0.8025	0.8684	0.8959
10	0.3333	0.5556	0.7037	0.8025	0.8683	0.9190
11	0.3333	0.5556	0.7037	0.8025	0.8683	0.9370
12	0.3333	0.5556	0.7037	0.8025	0.8683	0.9510

TABLE V

AVERAGE TOTAL EXECUTION TIME FOR PLANNING.

Scene (grid size)	Create Grid	Search	Total (msec)
Maze (50 x 50)	0.9	0.7	1.7
Maze (100 x 100)	3.4	3.7	7.2
Maze (150 x 150)	8.3	7.9	16.3
Office (45 x 45)	4.8	1.4	6.2
Office (90 x 90)	6.3	2.7	9.0
Office (135 x 135)	13.9	5.7	19.6

The average projection, search, and total *elapsed* execution times during repeated invocations of the planner during an interactive session were tabulated. The timing results are summarized in Table V. All values listed in the table are in units of milliseconds, and were averaged from N = 100 independent trials with varying goal locations and obstacle positions. Different grid resolutions were tested ranging between 45 and 150 cells on a side. The total number of triangle primitives in the Maze scene and the Office scene were 2,780 and 15,320 respectively. The start and goal locations used in these trials were specifically designed to force a majority of cells in the grid to be examined. Simpler path planning queries will produce proportionally faster path search times.

VII. SUMMARY AND DISCUSSION

We have presented a general method to improve the efficiency of search on Euclidean-cost grids and lattices. The key idea is to reduce the number of neighbor-node expansions by exploiting the properties of optimal paths as they relate to the overall structure of Euclidean-cost grids.

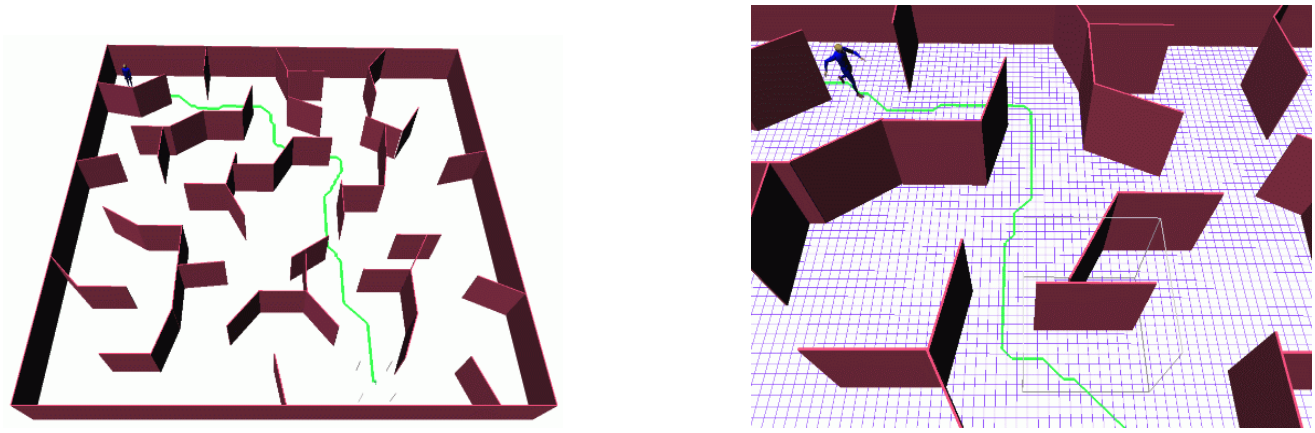


Fig. 5. A human figure navigating in a maze environment. The path is dynamically recomputed as the goal or obstacle locations change.

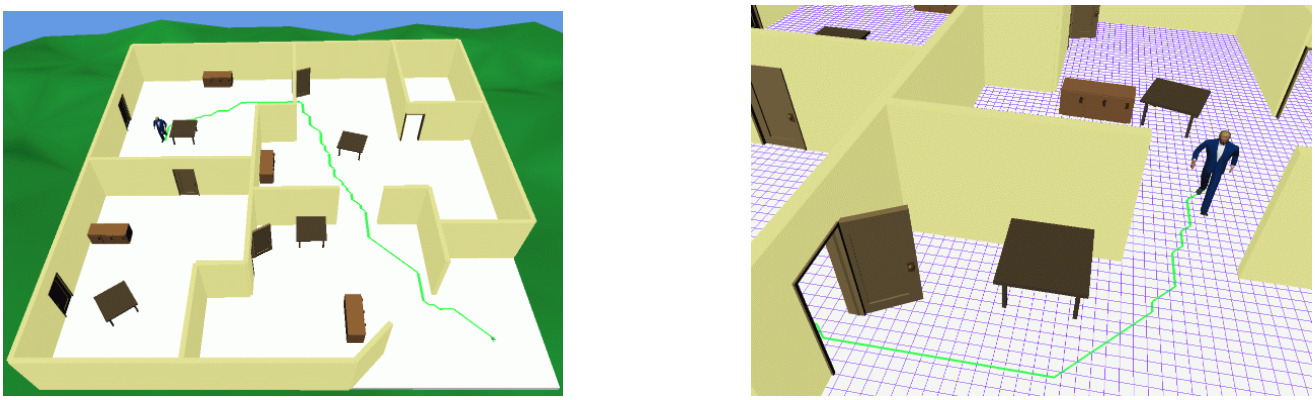


Fig. 6. A human figure navigating in an office environment. The left image shows the grid used for planning.

As with all other complete grid search methods, the search strategy implemented here an its basic form is impractical when dealing with large lattices, since the number of cells grows exponentially according to the size and dimension of the grid. For very large grids and lattices, data structures such as quadtrees, octrees, and their higher dimensional counterparts may be necessary to efficiently manage and store the free space. Addtional data structures could be created to limit the fine-grained path searching to a local area. There is a large body of work on finding optimal and approximately optimal paths in large networks (for a broad overview, see the survey by Mitchell [13]). For example, maximum distance cutoff values can potentially be used to define a local area. Goals outside the local area can then be mapped to the nearest free border cells in the grid. Alternatively, a multi-resolution hierarchical subdivision grid structure can potentially be used to first find a coarse path on the meta-grid, and then successively finer-grained paths in the sub-grids. Other possibilities include the use of intermediate goals defined by a global network of *grid landmarks* known to be connected by free paths.

Acknowledgment

This research was partially supported by NSF grants ECS-0325383, ECS-0326095, and ANI-0224419.

References

[1] S. M. LaValle and M. S. Branicky, "On the relationship between classical grid search and probabilistic roadmaps," in *Proc. Workshop on the Algorithmic Foundations of Robotics*, Dec. 2002.

[2] J. Kuffner, "Autonomous agents for real-time animation," Ph.D. dissertation, Stanford University, Stanford, CA, Dec. 1999.

[3] J. C. Latombe, *Robot Motion Planning*. Boston, MA: Kluwer Academic Publishers, 1991.

[4] J. H. Reif, "Complexity of the mover's problem and generalizations," in *Proc. 20th IEEE Symp. on Foundations of Computer Science (FOCS)*, 1979, pp. 421–427.

[5] J. T. Schwartz and M. Sharir, "On the 'piano movers' problem: Ii. general techniques for computing topological properties of real algebraic manifolds," *Advances in applied Mathematics*, vol. 4, pp. 298–351, 1983.

[6] J. Canny, *The Complexity of Robot Motion Planning*. Cambridge, MA: MIT Press, 1988.

[7] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[8] P. Hart, N. Nilsson, and B. Rafael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Sys. Sci. and Cyb.*, vol. 4, pp. 100–107, 1968.

[9] R. E. Bellman, *Dynamic Programming*. Princeton, New Jersey: Princeton University Press, 1957.

[10] D. Gelperin, "On the optimality of a*," *Artif. Intell.*, vol. 8, no. 1, pp. 69–76, 1977.

[11] M. Henzinger, P. Klein, and S. Rao, "Faster shortest-path algorithms for planar graphs," *J. Comput. Syst. Sci*, vol. 55, pp. 3–23, 1997.

[12] A. Stentz, "Optimal and efficient path planning for unknown and dynamic environments," *Int. Journal of Robotics and Automation*, vol. 10, no. 3, 1995.

[13] J. Mitchell, *Handbook of Computational Geometry*. Elsevier Science, 1998, ch. Geometric Shortest Paths and Network Optimization.