The 2009 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 11-15, 2009 St. Louis, USA

Extracting Paths From Fields Built With Linear Interpolation

Michael W. Otte and Greg Grudic

Abstract— **Algorithms such as Field-D* [1] use linear interpolation to infer continuous fields of $costdistance$-to-goal, where $costdistance$ is cost integrated over distance. Traditionally, field values have been used as direct input to trajectory planners. In contrast, we focus on extracting a minimum $costdistance$ path between two points, given the continuous field. We identify a suboptimal phenomenon that occurs when standard path extraction techniques are used on linearly interpolated quantity-to-goal fields. The phenomenon causes paths to drift sideways toward their horizontal or vertical bounds, resulting in increased path length and unnecessary turns. We find that the sub-optimality is a mathematical consequence of the linear interpolation used to create the $costdistance$-to-goal field. We present a possible improvement that calculates path segment directions using an interpolation between the $costdistance$-to-goal gradient vectors, and perform a series of experiments comparing this method with the current state-of-the-art. We find that the proposed method can achieve a significant reduction in path length error, and we provide discussion and examples of when it should and should not be used.**

I. INTRODUCTION

Traditional graph-search techniques such as Dijkstra's, A*, and D* find an optimal path with respect to a graph representation of the world [2]–[4]. This representation often has a two dimensional 4- or 8-connected structure. Unfortunately, the graph structure *itself* can lead to optimal graph paths that are sub-optimal with respect to the real world. Movement must be broken into a combination of horizontal and vertical transitions in a 4-connected graph, or decomposed along multiples of 45 degrees in an 8-connected graph.

Many (equally) optimal paths may exist with respect to the graph. For instance, given a uniform map, a path that moves through a 4-connected (8-connected) graph horizontally as far as possible and then vertically (diagonally) will have the same cost as a path that alternates between vertical and horizontal (diagonal) movement—see Figure 1. The former path is suboptimal globally, while the latter is suboptimal on the scale of a few map grids. The local sub-optimality of the staircase-like path can be corrected with a simple local planner, so the staircase-like path is more desirable in practice. However, given the set of all optimal graph-paths, the task of finding the best with respect to the real world is infeasible. A common solution is to break cost ties by moving toward the goal; however, this fails when the goal is blocked by an obstacle—see Figure 2.

Algorithms have recently been developed that largely avoid the tie-breaking problem by operating in the continuous domain that envelopes the 4- or 8-connected graph neighborhood. Graph node values represent a discrete sampling over a continuous field of either the $costdistance$ (i.e. cost integrated over distance) or $time$ required to reach the goal. Field-D* [1] operates much like D* Lite [5] (a replanning version of A*), except that it calculates $costdistance$-to-goal for continuous points on a graph edge using a linear interpolation between the $costdistance$-to-goal of the edge's end-nodes. This allows paths to follow trajectories in the continuous domain. Fast-marching level-set methods propagate a wave front representing the $time$ required to reach the goal [6], [7]. Assuming $time$ exists as an additional dimension, the front expands in the direction of the $time$ gradient but is slowed by obstacles. $time$ gradient approximations at each node are calculated in the continuous domain given node values in a narrow band around the current wave front.

Michael W. Otte and Greg Grudic are with the Department of Computer Science, University of Colorado at Boulder, 430 UCB, Boulder, Colorado 80309-0430 michael.otte@colorado.edu

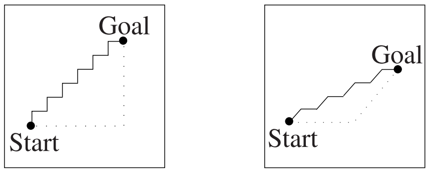

Fig. 1. Optimal paths of identical cost through uniform maps. The left and right maps use 4- and 8-connected graphs, respectively. The solid paths are more desirable than the dotted paths with respect to the real world.

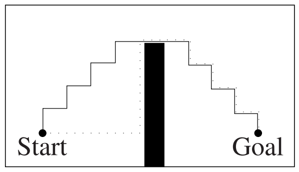

Fig. 2. Optimal paths of identical cost through a uniform map with an obstacle. The map uses a 4-connected graph. The dotted path breaks ties in cost by moving toward the goal, but the solid path is more desirable with respect to the real world.

Our paper is concerned with the extraction of paths between map start and goal locations, given a $costdistance$-to-goal field. The primary and most significant contribution of our work is the identification of, explanation of, and solution to an interesting suboptimal phenomenon that occurs during the path extraction process. The sub-optimality is an artifact of the linear interpolation used to create the field, and has not been previously addressed.

In practice, we use the Field-D* algorithm to build the cost field. However, our work is of relevance to any field that uses linear interpolation to estimate quantity-to-goal values in the continuous domain. Field-D* was originally developed to provide $costdistance$-to-goal values for a local trajectory planner [1]. As such, [1] is primarily concerned with the

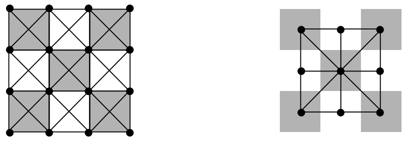

Fig. 3. Grid layout over map grids. Graph nodes and edges are black, map grids are gray and white. Field-D* places nodes at grid corners (left), while A*, D*, and D* Lite traditionally place nodes at grid centers (right).

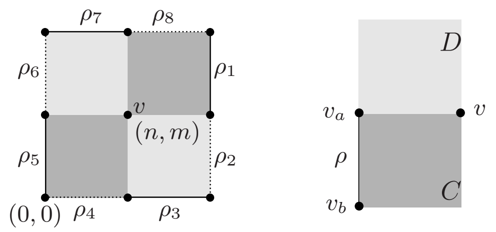

Fig. 4. (Left) The 8 edges used to determine the *costdistance*-to-goal of node v. (Right) Edge ρ connects nodes v_a and v_b. the *costdistance*-to-goal of v_a and v_b is g_a and g_b, respectively. C and D are map grids with *cost* c and d, respectively.

creation and maintenance of the field itself, and the extraction of complete paths is largely beyond its scope. A secondary contribution of our paper is to fill in the low-level details.

In Section II we provide a high-level description of how algorithms such as Field-D* populate the *costdistance*-to-goal field. This is also where the low-level details of basic path extraction are provided. In Section III we explain the suboptimal phenomenon and describe path extraction modifications aimed at combating it. In Section IV we perform a series of experiments to evaluate the effectiveness of our new techniques vs. prior methods, and discuss their strengths and limitations. Conclusions are given in Section V.

II. BACKGROUND

A. Field Creation

We now describe how linear interpolation is used to create a *costdistance*-to-goal field. We only cover details relevant to the sub-optimality and path extraction techniques presented in Section III. A complete algorithmic description of field generation and modification can be found in the original Field-D* papers by Ferguson and Stentz [1], [8].

As with D* [4] and D* Lite [5], a heuristic-based best-first heap is used to guide exploration from a goal node to a start node. The latter represents the robot's current position, and search occurs in the reverse direction of standard Dijkstra's [2] and A* [3]. When a node is popped off the top of the heap—or *expanded*—its unexpanded neighbors are added to (or updated within) the heap using a key based on their actual *costdistance*-to-goal plus a heuristic estimate of their *costdistance*-to-start. We assume the heuristic is admissible. Specifically, we use Euclidean $distance \geq 0$ to estimate $costdistance \geq 0$, assuming $cost > 0$. Recall that *costdistance* is *cost* integrated over *distance*.

Unlike D* and D* Lite, graph nodes are placed at the corners of map grids instead of at their centers (Figure 3-left vs. 3-right, respectively). An 8-connected graph structure is

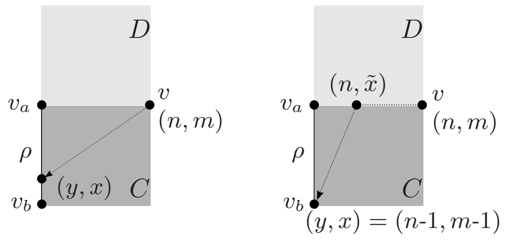

Fig. 5. Two possible ways to get from v to a point on edge ρ. (Left) The path goes directly to (y,x) through grid C. (Right) The path goes from (n,m) to $(n,\tilde{x})$ along the bottom of grid D, and then through grid C to $(y,x) = (n-1, m-1)$ at node v_b.

used to define neighboring nodes; however, travel through the map is not restricted to graph edges. Linear interpolation is used to determine the *costdistance*-to-goal for points along grid boundaries, based on the *costdistance*-to-goal of the corresponding horizontal or vertical edge's two end-nodes. Let (n,m) be the position of node v relative to the bottom-left node (Figure 4-left). Nodes are spaced 1 unit apart vertically and horizontally. Let (y,x) be a point along one of the 8 edges shown in Figure 4-left. $(y,x) \in \rho_i$ for $i = 1 \ldots 8$, and (y,x) exists in the same continuous coordinate space as (n,m). Let $g_{(n,m)}$ represent the *costdistance*-to-goal of v. When v is expanded, $g_{(n,m)}$ is calculated as follows:

$$g_{(n,m)} = \min\left([(n,m) \rightarrow (y,x)] + g_{(y,x)}\right)$$

where $[(n,m) \rightarrow (y,x)]$ is the *costdistance* of moving from (n,m) to (y,x) and $g_{(y,x)}$ is the linearly interpolated *costdistance*-to-goal of (y,x). There are 8 edges that must be examined to determine (y,x) and $g_{(n,m)}$. Without loss of generality, we restrict our discussion to finding the minimum *costdistance*-to-goal given a single edge (Figure 4-right). Similar calculations are performed for the remaining 7 edges and the minimum $g_{(n,m)}$ over all 8 is used as the final result.

Let c and d represent the map cost of grids C and D, respectively, and let g_a and g_b be the *costdistance*-to-goal of nodes v_a and v_b, respectively. [1] shows two ways a minimum *costdistance*-to-goal path may travel from v to (y,x) on edge ρ. These are illustrated in Figures 5-left and 5-right and described by Equations 2 and 4, respectively. Travel directly from v to v_a along the bottom of grid D is handled during the consideration of the edge above ρ.

$$g(y) = c\sqrt{1+(n-y)^2} + g_a(1-n+y) + g_b(n-y) \quad (1)$$

$$g_{(n,m)} = \min_{n-1 \leq y \leq n} g(y) \quad (2)$$

.

$$g(\tilde{x}) = d\,(m-\tilde{x}) + c\sqrt{1+(\tilde{x}-x)^2} + g_b \quad (3)$$

$$g_{(n,m)} = \min_{m-1 \leq \tilde{x} \leq m} g(\tilde{x}) \quad (4)$$

The minimum of Equations 2 and 4 is used as the final result with respect to edge ρ. The lesser of the two depends on the specific values of c, d, g_a, and g_b. Given the case illustrated in Figure 5-right, it is proven in [1] that the path will exit grid C at node v_b.

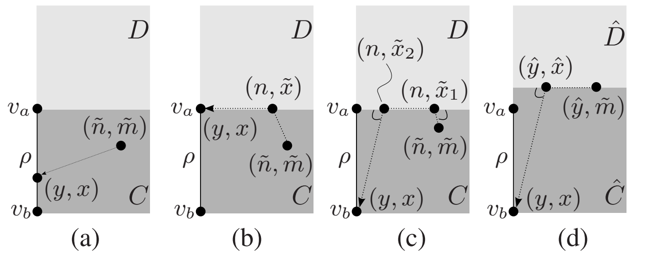

Fig. 6. The three possible ways to get from $(\tilde{n},\tilde{m})$ to a point on edge ρ. (a) The path goes directly to (y,x) through grid C. (b) The path goes from $(\tilde{n},\tilde{m})$ to $(n,\tilde{x})$ and then along the bottom of grid D to v_a. (c) The path goes from $(\tilde{n},\tilde{m})$ to $(n,\tilde{x}_1)$ then along the bottom of grid D to $(n,\tilde{x}_2)$ before cutting back across grid C to v_b. (d) A quantitatively identical case to (c), with respect to $costdistance$-to-goal.

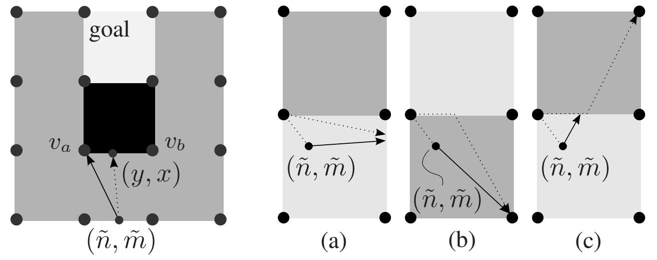

Fig. 7. Light to dark grids represent low to high cost, respectively. (Left) The point (y,x) is the next path point after $(\tilde{n},\tilde{m})$ using basic path extraction, dotted line. This is suboptimal because $costdistance$-to-gaol values at nodes v_a and v_b ignore the obstacle between them; thus, linear interpolation also ignores the obstacle. A better path goes directly to v_a, solid line. (Right) Three possible ways a path can reenter the bottom cell after transitioning to the upper left corner, dotted line. In each case, there is a less expensive path directly to the boundary edge, solid line.

B. Basic Path Extraction

A heuristic is used to focus field generation toward the robot. When robot position does not coincide with a grid corner, we must ensure field propagation to all nodes neighboring the robot (4 when the robot is not on a grid boundary and 6 when the it is on an edge but not a corner). Using Field-D*, this is achieved by 'pretending' the robot moves successively between the neighboring nodes, replanning at each. The extra replanning steps tend to execute quickly because $costdistance$-to-goal of neighboring nodes differs only by the $costdistance$ through the shared map grid.

Given a $costdistance$-to-goal field, a path between the robot and goal is iteratively extracted by finding the minimum $costdistance$-to-goal point on a nearby edge. Equations 1 through 4 must be modified to reflect the fact that the robot can exist at any point within a map grid. Equations 6 and 8 correspond to the cases illustrated in Figures 6-a and 6–b, respectively, and Equation 10 corresponds to the cases shown in Figures 6-c and 6-d, respectively.

$$g(y)=c\sqrt{(\tilde{m}-x)^2+(\tilde{n}-y)^2}+g_a(1-n+y)+g_b(n-y) \quad (5)$$

$$g_{(\tilde{n},\tilde{m})}=\min_{n-1\le y\le n} g(y) \quad (6)$$

where $x=m-1$.

$$g(\tilde{x})=c\sqrt{(\tilde{m}-\tilde{x})^2+(n-\tilde{n})^2}+d\,(\tilde{x}-x)+g_a \quad (7)$$

$$g_{(\tilde{n},\tilde{m})}=\min_{m-1\le\tilde{x}\le m} g(\tilde{x}) \quad (8)$$

where $x=m-1$.

$$g(\hat{x})=d(\tilde{m}-\hat{x})+c\sqrt{(\hat{x}-x)^2+(\hat{y}-y)^2}+g_b \quad (9)$$

$$g_{(\tilde{n},\tilde{m})}=\min_{m-1\le\hat{x}\le m} g(\hat{x}) \quad (10)$$

where $(y,x)=(n-1,m-1)$ and $\tilde{m}-\hat{x}=\tilde{x}_1-\tilde{x}_1$ and $\hat{y}=2n-\tilde{n}$. Grids $\hat{C}$ and $\hat{D}$ have cost c and d, respectively.

The $costdistance$-to-goal of the paths in Figures 6-c and 6–d are quantitatively similar. The latter is created from the former by removing the first two path segments, $(\tilde{n},\tilde{m})\rightarrow(n,\tilde{x}_1)\rightarrow(n,\tilde{x}_2)$, flipping them horizontally, and reattaching them to the third segment. The acute angles between the bottom of grid D and the diagonal path segments in Figure 6-c are equal to the acute angle between the bottom of grid $\hat{D}$ and the diagonal path segment in Figure 6-d.

Proof (by contradiction): Assume two different acute angles between the bottom of grid D and the diagonal path segments in Figure 6-c. When Figure 6-d is created (as described above) the portion of the minimum $costdistance$ path from $(\hat{y},\hat{x})$ to (y,x) will consist of two segments joined at an angle $\neq 0$ degrees. $costdistance$ obeys the triangle inequality because $c>0$ and distance is Euclidean. Therefore, a better path is found by using a single segment from $(\hat{y},\hat{x})$ to (y,x). This provides the necessary contradiction. $\square$

If $(\tilde{n},\tilde{m})$ exists on the grid boundary between C and D then the case illustrated in Figure 6-b can be ignored. If $(\tilde{n},\tilde{m})$ exists inside a grid (i.e. $\tilde{n}\neq\lfloor\tilde{n}\rfloor$ or $\tilde{m}\neq\lfloor\tilde{m}\rfloor$) then paths from $(\tilde{n},\tilde{m})$ to edge ρ may involve the grid below C instead of grid D. This is accomplished by substituting the appropriate coordinates into Equations 7 through 10.

Another consequence of heuristic focused field generation is the possibility that nodes in Equations 1-10 may not have been updated to reflect new cost-map changes (or even expanded). Assuming nodes have a default $costdistance$-to-goal of infinity, both cases are detected when the $costdistance$-to-goal of neighboring nodes differs by more than the minimum $costdistance$ through their shared map grid(s)—1, 1, and $\sqrt{2}$ times the map grid cost value for vertical, horizontal, and diagonal neighbors, respectively. Inconsistent nodes can either be ignored or temporarily reset to consistency by decreasing the larger $costdistance$-to-goal value appropriately. We use the latter method.

C. 1-Step Look-Ahead Path Extraction

In [1] Ferguson and Stentz give two ways to improve basic path extraction. The first is by using a 1-step look-ahead. Once the 'best' boundary point (y_{bst},x_{bst}) is found, its $costdistance$-to-goal is explicitly calculated by substituting $(y_{bst},x_{bst})=(\tilde{n},\tilde{m})$ in Equations 1 through 10. The resulting minimum $costdistance$-to-goal replaces the linearly interpolated $costdistance$-to-goal value used in the original calculation of (y_{bst},x_{bst}). Finally, (y_{bst},x_{bst}) is only used as the next path point if the updated $costdistance$-to-goal of the robot through (y_{bst},x_{bst}) is still less than going through any

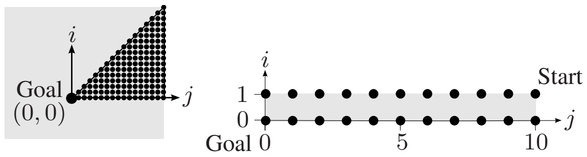

Fig. 8. (Left) The portion of the map we are considering, textured. (Right) The bottom two rows of nodes. The robot starts at $(1, 10)$ and the goal is at $(0, 0)$.

other candidate boundary point. This eliminates problems that occur due to linear interpolation between the edges of an obstacle, such as the case illustrated in Figure 7-left.

The second suggestion is to forbid movement to a corner (from the interior of a grid) if the next move is back through the original grid (Figure 7-right). Evaluating corner points with the 1-step look-ahead strategy will eliminate this type of error given Euclidean $distance \geq 0$ and $cost > 0$. The added cost of going back through the grid will be reflected in the updated $costdistance$-to-goal of the corner point. Paths that skip the corner point and move directly to the final edge will be less expensive, due to the triangle inequality.

We use a priority heap to keep track of the $costdistance$-to-goal of $(\tilde{n}, \tilde{m})$ for all relevant routes through nearby edges. The heap is initialized with $costdistance$-to-goal values from linear interpolation (Equations 1 through 10), along with the points used in their calculation. Next, we pop the 'best' value/point off the heap and update its value using 1-step look-ahead on the first point of the corresponding path segment (that is, (y, x), $(n, \tilde{x})$, (y, x), $(n, \tilde{x})$, or $(n, \tilde{x}_1)$ if the minimum $costdistance$-to-goal of $(\tilde{n}, \tilde{m})$ was calculated using Equation 2, 4, 6, 8, or 10, respectively). If the updated $costdistance$-to-goal through that point is greater than the new top-heap value, the point is reinserted into the heap using the updated value. This is repeated until the updated $costdistance$-to-goal of a popped point is less than the top-heap value (or the heap is empty), yielding the appropriate next point in the path. In the worst-case, every candidate point/value needs to be inserted twice.

III. IMPROVEMENT

A. The Problem

Consider a map of uniform cost $c > 0$, and let map grids be length 1. Let i and j denote the position of a node relative to the goal in the vertical and horizontal directions, respectively. Due to symmetry we can restrict our discussion to the part of the map between 0 and 45 degrees (Figure 8-left). The $costdistance$-to-goal of a node along the j axis is cj because it depends only on the $costdistance$-to-goal of a left neighbor plus the $costdistance$ along the horizontal length of a grid. Similarly, the $costdistance$-to-goal at nodes along the diagonal is given by $cj\sqrt{2}$ because it depends on the lower left neighbor plus the $costdistance$ along the diagonal of a grid. For now, we limit our discussion to the bottom two rows of nodes, and assume the robot starts at position $(i, j) = (1, 10)$. This is illustrated in Figure 8-right.

The $costdistance$-to-goal values of nodes $(1, j)$ for $j \geq 2$ are calculated using linear interpolation between their left and lower left neighbors, $(1, j-1)$ and $(0, j-1)$, respectively. Equation 2 is used because all map cost values are equal. Figure 9-a displays a back pointer from each node $(1, j)$ to the point $(y, x) = (y, j-1)$ at which Equation 1 reaches its minimum value (for $0 \leq y \leq 1$)). Figure 9-b superimposes the path that is extracted using the 1-step look ahead process described in Section II-C. Notice that the path is drawn down at an increasing rate until it reaches the j axis—at which point it goes straight to the goal. This is clearly sub-optimal. Given a uniform map, the least $costdistance$ path between robot and goal is along a straight line (Figure 9-c).

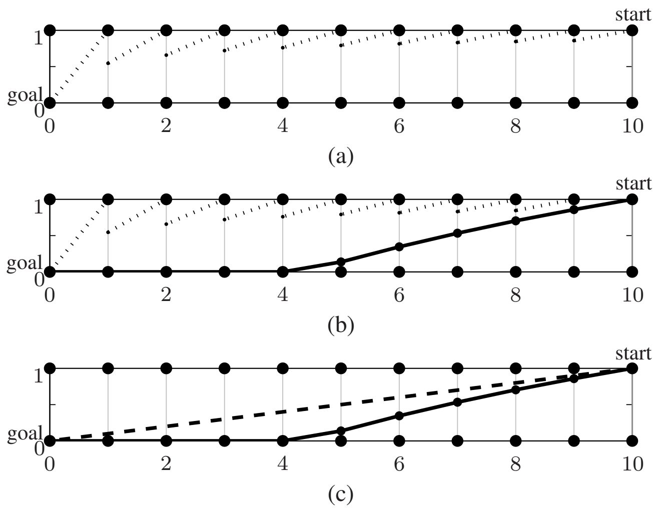

Fig. 9. Back-pointers used to determine $costdistance$-to-goal (dotted lines)—note that backpointers for the bottom row all point directly left. The path extracted with 1-step look-ahead (solid black line), and the optimal path found using prior knowledge (dashed line).

This phenomenon is explained by considering how y is calculated in Equations 2 and 6. For simplicity, we consider the latter, noting that Equations 2 is derived from Equation 6 by substituting $(n, m) = (\tilde{n}, \tilde{m})$. The boundary point through which we achieve the minimum $costdistance$-to-goal is found by taking the derivative of Equation 2, setting it equal to 0, and solving for y.

$$g(y) = c\sqrt{(\tilde{n} - y)^2 + (\tilde{m} - x)^2} + g_a(1 - n + y) + g_b(n - y)$$

$$g'(y) = 0 = \frac{c(y - \tilde{n})}{\sqrt{(\tilde{n} - y)^2 + (\tilde{m} - x)^2}} + g_a - g_b$$

$$\left(\frac{c}{g_b - g_a}\right)^2 - 1 = \frac{(\tilde{m} - x)^2}{(\tilde{n} - y)^2}$$

where $\tilde{m} \geq x$ in this part of the map.

$$y = \tilde{n} \pm \frac{\tilde{m} - x}{\sqrt{\left(c/(g_b - g_a)\right)^2 - 1}} \qquad (11)$$

We only need to consider the subtractive case of Equation 11 because $g_a \geq g_b$ and $\tilde{n} \geq y$ in this part of the map, and $0 \leq |g_b - g_a| \leq c$ given a uniform map. Due to the constraints of Equation 6, y is reset to $n - 1$ if $y < n - 1$ (this happens at position $j = 4$ in Figure 9-b).

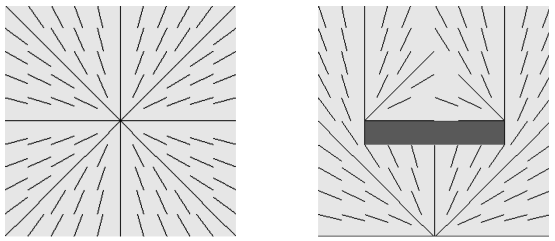

Fig. 10. Back-pointers used to determine $costdistance$-to-goal. (Left) The goal is in the center of a uniform map. (Right) The goal is in the bottom center and there is an obstacle in the center of the map.

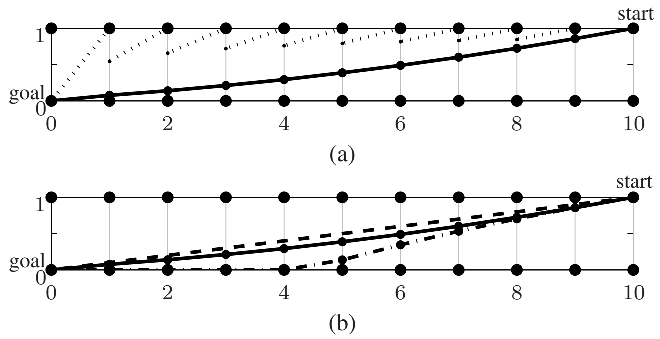

Fig. 11. Backpointers used to determine $costdistance$-to-goal (dotted lines)—note that backpointers for the bottom row all point directly left, the path extracted using our gradient interpolation method (solid black line), the path extracted with 1-step look-ahead (dot-dash line), and the optimal path found using prior knowledge (dashed line).

Insight is gained from the manipulation of Equation 11:

$$\frac{x-\tilde{m}}{y-\tilde{n}} = \sqrt{\left(c/(g_b-g_a)\right)^2-1} \tag{12}$$

Equation 12 shows that the slope of a path segment from $(\tilde{n},\tilde{m})$ to (y,x) is constant, given $|g_b-g_a|$ and c. Thus, the 1-step look-ahead path segment has the same slope as the backpointer from $(n,m)=(\lceil\tilde{n}\rceil,\lceil\tilde{m}\rceil)$. The path is pulled downward at an increasing rate as a consequence of changing backpointer slope with j. This can be seen in Figure 9-b. Once the path drops to $n-1$, the relevant backpointer changes—this is why the path moves straight to the goal after reaching the horizontal axis.

If $\tilde{n}>1$ then (x,y) will transition through the bottom of a grid whenever y is reset to $n-1$, but the overall behavior is the same. Near the 45 degrees axis, $|g_b-g_a|$ causes paths to be drawn toward the 45 degree axis instead of the horizontal axis. Near 22.5 degrees, the impetuses to move toward the horizontal and 45 degree axes balance, and suboptimal behavior does not occur. Figure 10-left shows backpointers for all nodes when the goal is located in the center of a uniform map.

B. A Possible Solution

By definition, node backpointers point in the same direction as $costdistance$-to-goal gradient vectors. This observation inspires an alternative path extraction technique. That is, use linear interpolation between gradient vector directions (i.e. backpointers) to determine path segment trajectories. This has the desirable effect of focusing path movement between neighboring backpointers (Figure 11-a). Although our method does not produce the optimal straight line path found with prior knowledge, it is often much closer to optimal than 1-step look-ahead. For instance, in Figure 11-b path length error is reduced by more than a factor of 10.

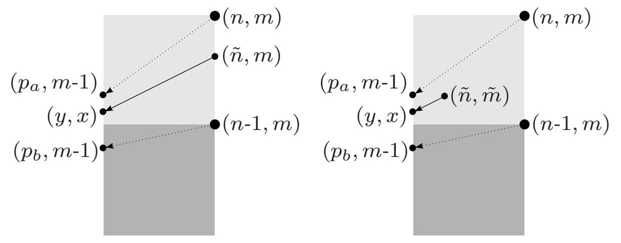

Fig. 12. Points used in our gradient direction interpolation method. The robot starts at point $(\tilde{n},m)$ (left), and $(\tilde{n},\tilde{m})$ (right). Backpointers from (n,m) and $(n\text{-}1,m)$ point to $(p_a,m\text{-}1)$ and $(p_b,m\text{-}1)$, respectively, appear dotted. The next point in the path is calculated as $(y,x)=(y,m\text{-}1)$.

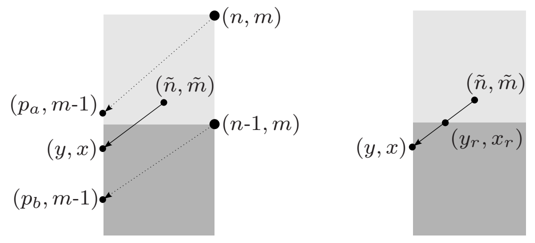

Fig. 13. When $y<n-1$, the next path coordinate is reset from (x,y) to (y_r,x_r). The latter is the intersection of the interpolated gradient direction with the grid boundary.

Gradient direction interpolation may not provide advantages if nearby gradients are parallel or diverging. Diverging backpointers happen in the presence of an obstacle (Figure 10-right), and interpolating between them will take the robot into the obstacle. Therefore, we only use the gradient interpolation when $(\tilde{n},\tilde{m})$ is between converging backpointers, defaulting to 1-step look-ahead otherwise. Similarly, we also default to 1-step look-ahead when a backpointer consists of more than one segment.

Consider the case illustrated in Figure 12-left. The robot starts on a vertical edge at $(\tilde{n},m)$ and the backpointers from nodes on either end of that edge converge—ending at $(p_a,m-1)$ and $(p_b,m-1)$, respectively. We interpolate between the backpointers to find the next path point $(y,x)=(y,m-1)$ as follows:

$$y = p_b + (p_a-p_b)(\tilde{n}-n+1) \tag{13}$$

In the case that the robot is not on an edge Equation 13 is modified as follows:

$$y = p_b + \frac{(p_a-p_b)\left(\tilde{n}-n+1+(n-1-p_b)(m-\tilde{m})\right)}{1+(p_a-p_b-1)(m-\tilde{m})}$$

This is illustrated in Figure 12-right, and can only happen during the calculation of the first path segment. In the event that $y<n-1$, the next path coordinate is reset to $(y_r,x_r)=(n-1,x_r)$ the intersection of the vector of travel with the horizontal grid boundary at $n-1$ (depicted in Figure 13).

$$x_r = \tilde{m}-(\tilde{m}-m+1)(\tilde{n}-n+1)/(\tilde{n}-y)$$

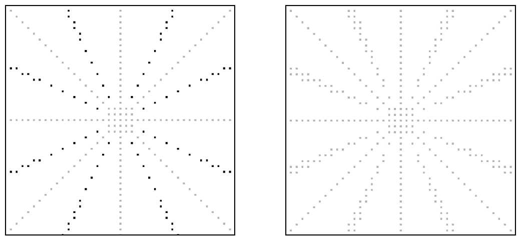

Fig. 14. A goal in the center of a uniform map. Dark nodes show where 1-step look-ahead is better than gradient interpolation. Light nodes show where both methods are the same. Nodes are omitted where gradient interpolation is better. (Left) Basic gradient interpolation. (Right) Defaulting to 1-step look-ahead for interpolated gradient angles between 20 and 30 degrees.

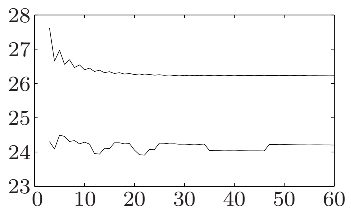

Fig. 15. The upper and lower bounds of gradient angles (vertical axis) that perform poorly when compared to 1-step look-ahead, plotted against column offset from goal (horizontal axis).

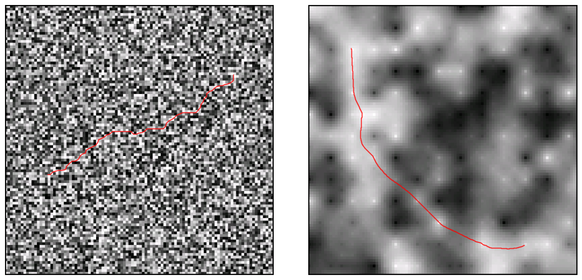

Fig. 16. Paths through maps from Experiment 1 (left) and Experiment 2 (right). Light to dark represents Low to high cost, respectively.

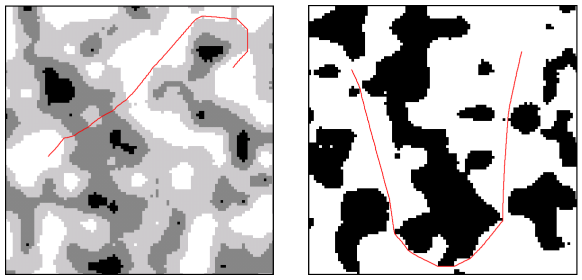

Fig. 17. Paths through maps from Experiment 3 (left) and Experiment 4 (right). Light to dark represents Low to high cost, respectively.

Figure 14-left shows nodes for which gradient interpolation does not outperform 1-step look-ahead, given a uniform map. The gradient method is sub-par when the robot starts around 22.5 degrees from a vertical or horizontal axis. We use bisection [9] to locate the lower and upper bounds of gradient angles associated with positive 1-step look-ahead performance. These results are plotted in Figure 15. The bounds appear to converge on 24.1 and 26.2 degrees, respectively, with lower and upper limits at 23.9 and 27.6 degrees, respectively. This suggests we should default to 1-step look-ahead when an interpolated gradient angle falls within this range. However, we find it beneficial to use a range of 20 to 30 due to interactions between the two methods. Figure 14-right compares the performance of this final modification to 1-step look-ahead, given a uniform map.

IV. EXPERIMENTS AND DISCUSSION

A. Experiments

Gradient interpolation is often better than 1-step look-ahead. However, there can exist locations in a random map where this is not the case. We hypothesize that, on average, gradient interpolation will outperform 1-step look-ahead, and conduct a series of experiments to test this hypothesis. We should reiterate that 1-step look-ahead is used in our gradient interpolation method whenever gradients do not converge. We also evaluate the utility of defaulting to 1-step look-ahead when interpolated gradient angles are between 20 and 30 degrees away from horizontal or vertical (henceforth we refer to this variation as *G.I. with angle check*). Finally, we test a combined technique that calculates paths using all three methods (1-step look-ahead, gradient interpolation, and G.I. with angle check) and then moves the robot based on whichever path has the minimum $costdistance$-to goal (henceforth this is referred to the as the *combined method*).

We experiment with four different types of randomly generated maps to determine if performance depends on the cost structure imposed on the environment. The maps used in Experiment 1 are created by randomly picking cost values out of a continuous uniform distribution between 1 and 10 (Figure 16-left). Experiment 2 uses fractal generated maps [10] with costs ranging from 1 to 10 (Figure 16-right). Maps in Experiment 3 are generated by thresholding fractal maps into 4 evenly separated cost classes, and then reassigning values of 1, 5, 10, and 20 in order of increasing threshold (Figure 19-left). Experiment 4 uses fractal maps that have been thresholded to contain two cost classes, these are then given values of 1 and 1000000 to represent free terrain and lethal obstacles, respectively (Figure 19-right).

100 trials per method per experiment are performed on both 100x100 and 400x400 sized maps. The goal and start locations are placed in columns $w/6$ and $4w/6$, respectively, where w is the width of the map. The start and goal rows are determined by the grids in those columns containing the minimum (pre-threshold) cost value. This ensures that the robot starts and ends in relatively safe terrain.

Given the uniform map in Figure 11-b, 1-step look-ahead produces paths that are 0.34% too expensive (i.e. they have relative $costdistance$ error of 0.34% when compared to the optimal path that is known *a priori*). This is quite good; however, gradient interpolation produces paths that are only 0.031% too expensive—a reduction in error of 91%. Thus, although 1-step look-ahead is close to the optimal solution, gradient interpolation is an order of magnitude closer. We would like to be able to perform a similar evaluation given

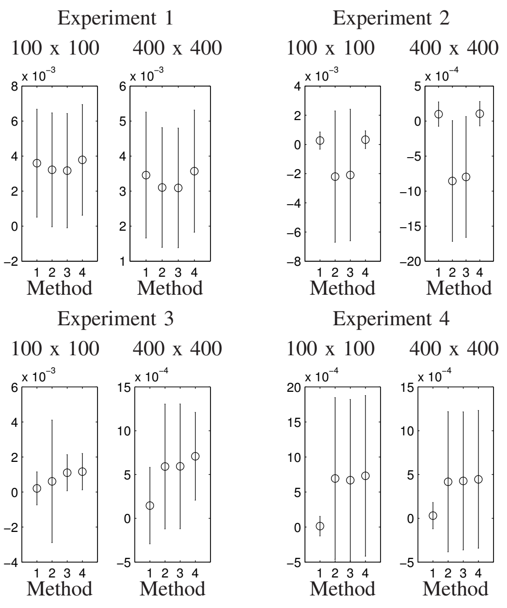

Fig. 18. Means and standard deviations of path length reductions relative to basic path extraction for Experiment 1 (top-left), Experiment 2 (top-right), Experiment 3 (bottom-left), and Experiment 4 (bottom-right). x−axis values indicate method as follows: (1) 1-step look-ahead, (2) gradient interpolation, (3) G.I. with angle check, (4) combined method.

TABLE I
RELATIVE ERROR REDUCTION FROM NAIVE PATH EXTRACTION: T-TEST P-VALUES OF 1-STEP LOOK-AHEAD VS. OTHER METHODS

Map Size: 100 x 100				
Method	Experiment			
	1	2	3	4
Gradient Interpolation (G.I.)	0.40	$< .001$	0.27	$< .001$
G.I. With Angle Check	0.34	$< .001$	$< .001$	$< .001$
Combined Method	0.67	0.46	$< .001$	$< .001$
Map Size: 400 x 400				
Method	Experiment			
	1	2	3	4
Gradient Interpolation (G.I.)	0.15	$< .001$	$< .001$	$< .001$
G.I. With Angle Check	0.14	$< .001$	$< .001$	$< .001$
Combined Method	0.66	.80	$< .001$	$< .001$

TABLE II
RELATIVE ERROR REDUCTION FROM NAIVE PATH EXTRACTION: RATIO OF OTHER METHODS TO 1-STEP LOOK-AHEAD

Map Size: 100 x 100				
Method	Experiment			
	1	2	3	4
Gradient Interpolation (G.I.)	.894	−8.35	2.90	50.5
G.I. With Angle Check	.881	−7.92	5.24	48.7
Combined Method	1.05	1.24	5.53	53.2
Map Size: 400 x 400				
Method	Experiment			
	1	2	3	4
Gradient Interpolation (G.I.)	.897	−8.65	4.08	13.7
G.I. With Angle Check	.894	−8.06	4.09	14.0
Combined Method	1.03	1.06	4.87	14.6

randomly generated maps. Unfortunately, this is prohibited by the fact that we do not know the theoretically optimal $costdistance$ path through a continuous random map. Instead, we look at the *relative error reduction* from the naive extraction technique presented in (Section II-B). Relative error reduction is calculated as follows:

$$\text{relative error reduction} = \frac{\text{naive}_{cstdst} - \text{other}_{cstdst}}{\text{naive}_{cstdst}}$$

where naive_{cstdst} and other_{cstdst} are the total $costdistance$ of a path extracted using the naive and comparison techniques, respectively. The mean relative error reduction over the 100 trials is plotted for each experiment in Figure 18, error-bars are plotted 1 standard deviation above and below the mean. p-values are presented in Table I for a student's t-test given the null-hypotheses that relative error reduction values observed using 1-step look-ahead are generated from the same normal distributions as those observed using the gradient interpolation methods.

The ratio between the mean relative error reduction of gradient interpolation methods to those of 1-step look-ahead are presented in Table II. For instance, in Experiment 4 on a 100x100 map, the average reduction in error obtained by using gradient interpolation instead of the naive method is 50.5 times greater than the average reduction in error obtained by using 1-step look-ahead instead of the naive method. Values greater than 1 indicate better performance than 1-step look-ahead, values between 0 and 1 indicate worse performance than 1-step look-ahead, and negative values indicate worse performance than the naive method.

To demonstrate our method on a real problem we perform a fifth experiment using geographical data from the United States Geological Survey (USGS)[1]. Elevation information is used to create a map of the Hawaiian islands at a resolution of 0.1565 kilometers per grid. Ocean and land are defined as free and obstacle, respectively, and water routes are calculated from Hilo Bay to four other bays/beaches. Relative error reductions vs. the naive method are presented in Table III. Raw scores are shown (instead of the ratio to 1-step look-ahead) because 1-step look-ahead has a relative error reduction of 0.

TABLE III
RELATIVE ERROR REDUCTION FROM NAIVE PATH EXTRACTION: RAW SCORES FROM HAWAII DATA

Method	Hilo Bay to :			$(\mathbf{x}10^{-5})$
	Kiholo	Hana	Kephuli	Waikiki
1-step look-ahead	0	0	0	0
Gradient Interpolation	16.84	4.10	1.21	2.51
G.I. With Angle Check	16.16	4.13	1.82	1.93

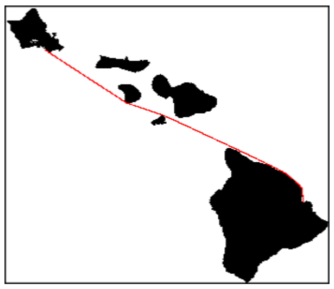

Fig. 19. Water route from Hilo Bay to Waikiki, Hawaii.

[1]http://seamless.usgs.gov/index.php

B. Discussion

We observe similar trends in all map sizes. Experiment 1 shows that all gradient interpolation based methods perform statistically similar to 1-step look-ahead on maps that are created randomly from a continuous uniform distribution. Despite the statistical similarity, we advise against using gradient interpolation on this type of map unless the combined method is used, given that the latter is the only technique providing any improvement over 1-step look-ahead. Experiment 2 illustrates that gradient interpolation methods should not be used on maps containing gently varying cost values (again, using the combined method provides a marginal advantage and will, at least, not handicap a system).

On the other hand, Experiments 3, 4, and 5 show environments in which gradient interpolation excels. Namely, maps with continuous regions of similar cost. Gradient interpolation has been developed in response to a suboptimal phenomenon observed on uniform maps; therefore, it is not surprising that it performs well in these cases. Binary ('obstacle' vs. 'free') maps and maps with a limited set of discrete cost regions are commonly used in practice. Also, continuous cost regions often occur in other maps as the result of two common algorithmic considerations: (1) Predefined values of unknown cost at unexplored regions of the environment. (2) Cost thresholding used to eliminate noise and other small variations that cause unnecessary meandering.

Restricting our discussion to environments where gradient interpolation performs well and enough trials exist to perform analysis (Experiments 3 and 4), G.I. with angle check does not statistically improve performance vs. normal gradient interpolation ($p > .05$). We also observe that larger maps tend to decrease the amount of improvement provided by gradient interpolation. That said, in all but one experiment/system combination, the improvement is statistically significant ($p < .001$) and provides relative error reduction > 4 times that of 1-step look-ahead vs. the naive method. Finally, we note that the combined method always provides better performance than any of the other methods. This is observed even in terrain where gradient interpolation does not tend to perform well, and explained by the fact that the combined method uses whichever technique provides the most advantage from a given location.

V. CONTRIBUTIONS AND CONCLUSIONS

The four main contributions of our paper include: (1) We identify a suboptimal phenomenon that occurs when standard path extraction techniques are used to find a complete path through a continuous $costdistance$-to-goal field that has been built using linear interpolation. The phenomenon causes paths to drift sideways toward their horizontal or vertical bounds such that path length is increased. A secondary consequence is that paths contain unnecessary turns at positions where a path meets its horizontal or vertical bound.

(2) We mathematically explain why this sub-optimality occurs. In previous techniques (naive method and 1-step look-ahead), all extracted path segments traveling directly through the same grid are parallel. This is suboptimal for points in the continuous space between nodes and is a consequence of using linear interpolation to calculate $costdistance$-to-goal values there.

(3) We present a possible solution where path segment directions are found using linear interpolation between the $costdistance$-to-goal gradient directions at neighboring nodes. A path segment that moves in a direction between the gradient directions of its neighboring nodes can significantly reduce overall path length error (over an order of magnitude for some paths through uniform maps).

(4) We perform a series of experiments to evaluate the utility of three gradient interpolation methods in four different types of maps and on two map sizes. We also demonstrate gradient interpolation outperforming 1-step look-ahead on four free vs. obstacle maps created from USGS elevation data.

Gradient interpolation provides significant reduction in path length error on maps containing continuous regions of identical cost, but little or no advantage on maps containing gradual cost transitions. Maps where gradient interpolation excels include: common binary ('obstacle' vs. 'free') maps, maps with a finite set of region based costs (e.g. 'dirt,' 'grass,' 'rock,' 'water,' etc.), and maps where a fixed cost for unknown terrain or cost thresholding is used. The more complicated G.I. with angle check does not provide significant improvement over gradient interpolation. The combined method uses gradient interpolation when doing so is advantageous and defaults to 1-step look-ahead otherwise; therefore, it is recommended when map type is unknown or ambiguous.

Although we have focused on path extraction from $costdistance$-to-goal fields created with Field-D* [1], our techniques and discussion are applicable to other continuous quantity-to-goal fields built using linear interpolation.

REFERENCES

[1] D. Ferguson and A. Stentz, "Using interpolation to improve path planning: The field D* algorithm," *Journal of Field Robotics*, vol. 23, pp. 79–101, 2006.

[2] E. W. Dijkstra, "A note on two problems in connection with graphs," in *Numererical Mathematics*, vol. 1, 1959, pp. 269–271.

[3] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," in *Proc. IEEE Transactions On System Science and Cybernetics (SSC-4)*, 1968, pp. 100–107.

[4] A. Stentz, "The focussed D* algorithm for real-time replanning," in *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1995.

[5] S. Koenig and M. Likhachev, "Improved fast replanning for robot navigation in unknown terrain," in *Proc. IEEE International Conference on Robotics and Automation (ICRA'02)*, 2002.

[6] J. A. Sethian, "A fast marching level-set method for monotonically advancing fronts," *Proc. Nat. Acad. Sci.*, vol. 93, pp. 1591–1595, 1996.

[7] R. Philippsen, *A light formulation of the E* interpolated path replanner*. Autonomous Systems Lab, Ecole Polytechnique Federale de Lausanne, 2006.

[8] D. Ferguson and A. Stentz, "Field D*: An interpolation-based path planner and replanner," 2005.

[9] D. Kincaid and W. Cheney, *Numerical Analysis*. Pacific Grove, CA, USA: Brooks/Cole, 2002.

[10] A. Fournier, D. Fussell, and L. Carpenter, "Computer rendering of stochastic models," *Communications of the ACM*, vol. 25, pp. 371–384, 1982.