Continuous Path Planning with Multiple Constraints*

Ian M. Mitchell† and Shankar Sastry
Department of Electrical Engineering & Computer Science
University of California, Berkeley
{imitchel,sastry}@eecs.berkeley.edu

August 27, 2003

A condensed version of this paper appears in the CDC 2003 proceedings.

Abstract

We examine the problem of planning a path through a low dimensional continuous state space subject to upper bounds on several additive cost metrics. For the single cost case, previously published research has proposed constructing the paths by gradient descent on a local minima free value function. This value function is the solution of the Eikonal partial differential equation, and efficient algorithms have been designed to compute it. In this paper we propose an auxiliary partial differential equation with which we can evaluate multiple additive cost metrics for paths which are generated by value functions; solving this auxiliary equation adds little more work to the value function computation. We then propose an algorithm which generates paths whose costs lie on the Pareto optimal surface for each possible destination location, and we can choose from these paths those which satisfy the constraints. The procedure is practical when the sum of the state space dimension and number of cost metrics is roughly six or below.

*Research supported by ONR under MURI contract N00014-02-1-0720.
†Corresponding author.

1 Introduction

Few problems are as well studied as the path planning or routing problem; it appears in engineering disciplines that vary from robotics to wireless communication to matrix factorization. A major challenge in developing solutions to the problem are the many, sometime subtle, variations it can adopt: the topology of the state space and cost metrics, the types of acceptable paths, the number of sources and destinations, the acceptable degree of optimality, etc. While every variant has at least one solution method—enumerate all feasible paths until an acceptably optimal one is found—the key to developing efficient solution algorithms is to take advantage of the particular properties of the variant of interest.

In this paper we examine the path planning problem in a continuous state space subject to constraints on additive path integral cost metrics. The original motivation for this work was the planning of fuel constrained flight paths for unmanned aerial vehicles through enviroments with varying levels of threat. Paths are generated by gradient descent on a value function (with no local minima), which is the solution of an Eikonal partial differential equation (PDE).* Path integral costs are evaluated by solving an auxiliary PDE. Both PDEs can be solved quickly for low dimensional systems, thus yielding a practical algorithm for path planning. Because both PDEs are solved over the entire state space, paths to any possible destination can be rapidly evaluated.

To handle constraints, we sample the Pareto optimal surface looking for paths with feasible combinations of costs. The sampling method only reaches the convex hull of the Pareto surface, so for nonconvex problems it may not always find the optimal feasible path; however, in our experience the degree of nonconvexity has not been enough to cause significant problems.

The asymptotic cost of the algorithm is $\mathcal{O}(MdN^d \log N)$, where M is the number of sampled points on the Pareto optimal surface, d is the state space dimension, and N is the number of grid points in each state space dimension. To adequately sample the Pareto surface, M will typically be exponential in the number of separate cost functions k. While these two exponentials are daunting, in practice the algorithms described below are quite practical on

*Classical applications of the Eikonal PDE are in the fields of optics and seismology. Its solution can be interpreted as a first arrival time or a cost to go, depending on whether the boundary conditions represent sources or sinks.

the desktop when the sum $k + d$ is less than around five or six; for example, section 3 includes a problem in two dimensions with three cost functions that is solved in less than one minute on the authors' laptop computer.

Gradient descent on a value function solution of the Eikonal equation has been used previously for unconstrained, single cost path planning problems. The innovative contribution of this paper is the application of auxiliary PDEs to calculate multiple path integral costs, and the use of those costs to find constrained optimal paths.

In the remainder of this section we formally outline our path planning problem and examine related work. Subsequent sections describe the algorithm, provide several examples, and discuss extensions to more general problems.

We work in a state space $\mathbb{R}^d$. Unless otherwise specified, norms are Euclidean $\|\cdot\| = \|\cdot\|_2$. Let $\mathbb{R}^+ = (0, \infty)$ be the set of strictly positive real numbers.

1.1 Problem Definition

A *path* $p : \mathbb{R}^+ \times \mathbb{R}^d \to \mathbb{R}^d$ is parameterized by an arclength $s \in \mathbb{R}^+$ and a destination location $x \in \mathbb{R}^d$. Assume that all paths have a single *source location* $x_s \in \mathbb{R}^d$ (we will relax this assumption later). The *path cost functions* $\{c_i(x)\}_{i=1}^k$, where $c_i : \mathbb{R}^d \to \mathbb{R}^+$, are continuous, bounded and strictly positive. The cost along a path is additive, so the total cost of a path can be evaluated by a *path integral*

$$P_i(x) = \int_0^T c_i(p(s,x))ds, \qquad \text{where } \begin{cases} p(0,x) &= x_s, \\ p(T,x) &= x. \end{cases} \tag{1}$$

In words, $P_i : \mathbb{R}^d \to \mathbb{R}$ is the total cost, according to path cost function $c_i(\cdot)$, of following the path $p(\cdot, x)$ from the source location x_s to the point x.

As an example, consider planning the flight path of an aircraft from its base at x_s to various destinations. The most obvious path cost function is fuel, which we approximate as a constant $c_{\text{fuel}}(x) = c_{\text{fuel}}$. A second path cost function might be the threat of inclement weather $c_{\text{weather}}(x)$. A third might be uncertainty about the enviroment, encoded as $c_{\text{uncertain}}(x)$. The latter two costs are inhomogenous, meaning that their value depends on x. Examples of cost functions are shown in figures 2 and 5.

There are two related problems that we might wish to solve starting from the parameters x_s and $\{c_i(x)\}_{i=1}^k$ described above. Given some set of *cost constraints* $\{C_i\}_{i=1}^k$, where $C_i \in \mathbb{R}^+$, we might want to find feasible paths such that $P_i(x) \leq C_i$ for all $i = 1 \ldots k$. Alternatively, we might try to minimize $P_1(x)$ subject to constraints on the remaining costs $P_i(x) \leq C_i$ for all $i = 2 \ldots k$. In either case, we will usually be interested in quantitative measures of the tradeoffs between the various path cost functions; for example, in the second type of problem what relaxation of the constraint C_2 would be required to cut the cost $P_1(x)$ in half?

1.2 Related Work

The significance of the most closely related algorithmic work [1, 2, 3] is discussed in section 2.4. However, similar problems have been investigated in several other fields.

Path planning is a central endeavor in robotics research [4], so we mention only the most closely related work. The algorithm discussed in this paper could be categorized as a potential field approach [5], in the sense that the paths are determined by gradient descent on a scalar function defined over the state space. In particular, the value function constructed in section 2 is an example of a navigation function [6]—a potential field free of the local minima that hinder most potential field methods (although in general it will contain saddle points). The specific use of the Eikonal equation for robot path planning in the single cost case was examined in [7], and is equivalent to the approach used in [8].

Independently, the networking community has been solving constrained shortest path planning on discrete graphs [9, 10, 11], primarily for the purpose of network routing. While this research involves problems with multiple costs, it makes the assumption that the number of distinct cost values possible at any node in the graph is finite and bounded. The resulting algorithms are pseudo-polynomial time: polynomial in the size of the graph and in the value of any constraints. If we seek a convergent approximation for the continuous path planning problem, we cannot assume that the value function can be discretized and thus we cannot use pseudo-polynomial time algorithms. It should be noted, however, that our method for exploring the Pareto optimal surface of possible path costs by sampling values of λ (see section 2.3) is

equivalent to the fastest algorithm proposed for finding constrained shortest paths in [11]. The distinction between their algorithm and ours is the underlying shortest path problem: discrete in their case, continuous in ours.

The related work that is closest mathematically is a tomographic application [12], which uses the Eikonal equation (2) to calculate travel time and a version of the path integral PDE (4) to determine perturbations of a linearized form of the Eikonal equation. To our knowledge, the use of (4) for evaluating path integral costs is original.

2 Value Function Solution

We discuss the value function method for finding the shortest path in the single cost case, and then how to compute path integrals along value function generated paths. With these tools we can explore the range of paths that might meet the constraints when multiple cost functions are involved. This section concludes with a discussion of an efficient algorithm for solving the required differential equations.

2.1 Single Objective Shortest Path

Consider the simplest case $k = 1$ with a single path cost function $c(x) = c_1(x)$ (because it will be used to generate a value function, we call this cost $c(x)$ the *value cost function*). It can be shown that the minimum cost to go from the source x_s to any point x in the state space is the solution of the inhomogenous Eikonal equation

$$\begin{aligned} \|\nabla V(x)\| &= c(x) \quad \text{for } x \in \mathbb{R}^d, \\ &\text{with boundary condition } V(x_s) = 0. \end{aligned} \tag{2}$$

The solution $V : \mathbb{R}^d \to \mathbb{R}$ of this PDE is called the *value function*. In practice V is rarely differentiable and therefore (2) does not have a solution in the classical sense. The viscosity solution [13] is the appropriate weak solution for the shortest path problem. In section 2.4 we shall discuss algorithms for computing accurate numerical approximations of the viscosity solution of (2), but for now the important fact is that efficient schemes exist for problems of reasonably low dimension.

Given the viscosity solution V, the optimal path $p^*(\cdot, x)$ can be determined by gradient descent of V from a fixed target location x. In practical terms, let $\hat{p}(s, x)$ be a path that starts at a particular x and terminates at x_s. Then $\hat{p}$ is the solution to the ordinary differential equation (ODE)

$$\begin{aligned} \frac{d\hat{p}(s,x)}{ds} &= \frac{\nabla V(\hat{p}(s,x))}{\|\nabla V(\hat{p}(s,x))\|} \quad \text{for } s \in \mathbb{R}^+ \text{ and fixed } x \in \mathbb{R}^d, \\ &\text{with initial condition } \hat{p}(0,x) = x. \end{aligned} \tag{3}$$

We stop extending the solution at some $\hat{s}$ such that $\hat{p}(\hat{s}, x) = x_s$. Then $\hat{s} = T$ is the arclength of the shortest path from x_s to x, and that path is given by $p^*(s, x) = \hat{p}(T - s, x)$. Because $V(x)$ is the cost to get to x from x_s along path p^*, the path integral for this path is $P^*(x) = V(x)$. The gradient descent (3) cannot get stuck in local minima because V has none.[†] In theory, (3) can get stuck at saddle points of V, but the stable manifolds of such points are of measure zero in the state space, and are thus unlikely to be a problem in practical implementations subject to floating point roundoff noise.

2.2 Computing Path Integrals

Throughout the remainder of this paper, we consider only paths generated by (3) for some value function V. In this section we examine how to compute the path integral when the value cost function is not the same as the path cost function. To differentiate the two cost functions, we denote the value cost function in (2) by $c(x)$ and the path cost function in (1) by $c_i(x)$. Both must use the same source location x_s.

Starting from the differential form of (1), we formally derive a PDE for the path integral $P_i(x)$

$$\begin{aligned} \frac{dP_i(p(s,x))}{ds} &= c_i(p(s,x)), \\ \frac{\partial P_i(p(s,x))}{\partial p(s,x)} \cdot \frac{dp(s,x)}{ds} &= c_i(p(s,x)), \\ \nabla P_i(p(s,x)) \cdot \frac{\nabla V(p(s,x))}{\|\nabla V(p(s,x))\|} &= c_i(p(s,x)), \\ \nabla P_i(p(s,x)) \cdot \nabla V(p(s,x)) &= c_i(p(s,x)) c(p(s,x)), \end{aligned}$$

[†]Easily seen if V is differentiable, since a local minimum would require $\nabla V(x) = 0$, but $c(x) > 0$. A more rigorous argument based on the positivity of c can be constructed when V is a viscosity solution.

where (3) is used in the second step and (2) is used in the third. Consequently, for all reachable points in the state space,

$$\begin{aligned} \nabla P_i(x) \cdot \nabla V(x) &= c_i(x)c(x) \quad \text{for } x \in \mathbb{R}^d, \\ &\text{with boundary condition } P_i(x_s) = 0. \end{aligned} \tag{4}$$

Because the cost structure is isotropic (independent of path direction) the system is small time controllable and for our single source version all states will be reachable. The derivation above assumes that all the functions involved are differentiable, but as was stated earlier this assumption will fail for $V(x)$ and therefore likely also for $P_i(x)$. We are in the process of developing a robust proof that the viscosity solution of (4) is the path cost integral we seek.

When solving (4), $P_i(x)$ is the unknown while $V(x)$, $c_i(x)$ and $c(x)$ are all known. Not surprisingly, (2) can be recovered from (4) for the single cost case of the previous section by substituting $c_i(x) = c(x)$ and $P_i(x) = V(x)$.

2.3 Exploring Potential Paths

As discussed in section 1.1, one of our goals was an algorithm to generate feasible paths subject to a collection of cost constraints. In the previous two sections we described PDEs whose solutions were a path generating value function V in (2) and the path integrals P_i for those paths in (4). The remaining missing ingredient is the value cost function $c(x)$ in (2). In this section we discuss the results of using convex combinations of the path cost functions as the value cost function.

We start with the simplest multiobjective case, $k = 2$. Let

$$c^\lambda(x) = \lambda c_1(x) + (1 - \lambda)c_2(x) \quad \text{for some } \lambda \in [0, 1].$$

Then evaluate (2) and (4) for $V^\lambda(x)$, $P_1^\lambda(x)$ and $P_2^\lambda(x)$. The first thing to notice is that $\lambda = 1$ calculates paths optimal in c_1 and $\lambda = 0$ paths optimal in c_2. Therefore, if $P_1^{\lambda=1}(x) > C_1$ or $P_2^{\lambda=0}(x) > C_2$ for some point x, there cannot be any feasible paths from x_s to x. Intermediate values of λ will generate paths lying somewhere between these two extremes.

Testing all possible values of λ would effectively construct the convex hull of the Pareto optimal tradeoff curve between the two cost functions. Figure 1 shows a possible Pareto curve for a single point x, the points on that curve

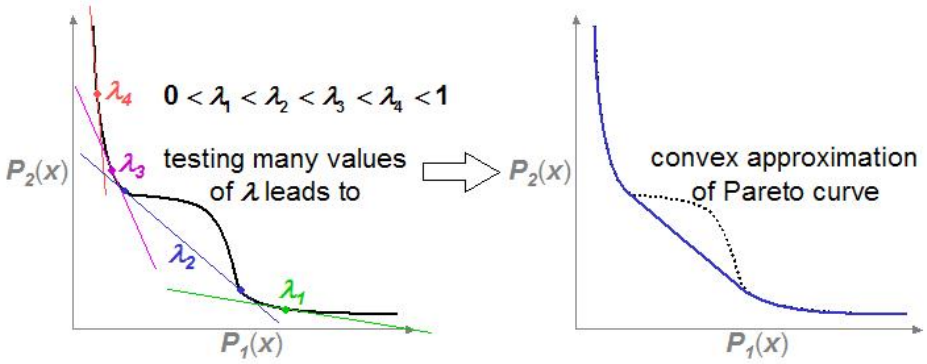

Figure 1: Pareto optimal curve for a particular destination state x. Left: each value of λ samples a point on the curve. Right: testing all values of λ would yield a convex approximation of the Pareto curve.

determined by several values of λ, and the convex hull of that curve. A point on the curve is a pair $(P_1^\lambda(x), P_2^\lambda(x))$ and lies where a line of slope $\frac{\lambda}{(\lambda-1)}$ is tangent to the Pareto curve. Therefore, λ is a quantitative measure of the tradeoff between the two cost functions.

In general the Pareto curve is not convex, so this method may fail to detect a feasible path even if one exists. Nonconvexity in the Pareto curve will manifest itself by jumps in the values of the path integrals $P_1^\lambda(x)$ and $P_2^\lambda(x)$ for fixed x as λ is varied continuously; for example, consider the jump in path integrals as λ is varied in the range $[\lambda_2-\epsilon, \lambda_2+\epsilon]$ for some small $\epsilon > 0$ in figure 1. However, neighboring values of λ can be used to bound the error in the convex approximation and nonconvexity has not been a problem in our experience. It should be pointed out that the Pareto curve characterised above is for a single point x in the state space. Because V^λ and P_i^λ are calculated over the entire state space, the technique actually approximates a separate Pareto curve for all points x.

To handle the case $k > 2$, we simply choose a set $\{\lambda_j\}_{j=1}^k$ such that $\lambda_j \in [0, 1]$ for all j and $\sum_{j=1}^k \lambda_j = 1$. Then $c^{\{\lambda_j\}}(x) = \sum_{j=1}^k \lambda_j c_j(x)$, and we can solve for the corresponding value function and path cost integrals. In this case it is the convex hull of the Pareto optimal surface that is explored as the set $\{\lambda_j\}$ is varied.

2.4 Numerical Algorithms

The discussion above would be nothing more than a mathematical diversion if it were not possible to solve (2), (3) and (4) numerically for some practical problems. In this section we briefly outline an existing efficient algorithm for solving (2), and modify that algorithm slightly to handle (4) as well. We postpone implementation details to section 3.4.

To treat (3), we assume that (2) and (4) can be computed for a variety of λ values to generate $V^\lambda(x)$ and $\{P_i^\lambda(x)\}_{i=1}^k$. Then a particular $\hat{\lambda}$ is chosen such that any path integral constraints are satisfied ($P_i^{\hat{\lambda}}(x) \leq C_i$). A path is determined by solving (3) for value function $V^{\hat{\lambda}}(x)$ with a standard ODE integration method, such as Runge-Kutta.

Solving (2) efficiently relies on an algorithm first described in [1], although the explanation that follows is based on an independently developed equivalent version [2] commonly known as the *Fast Marching Method* (FMM). This algorithm is basically the Dijkstra algorithm for computing shortest paths in a discrete graph [14], suitably modified to deal with a continuous state space. For readers interested in alternatives, there are other algorithms for solving (2); for example, [15, 16].

The value function $V(x)$ is approximated on a Cartesian grid over the state space with N nodes in each dimension, for a total of N^d nodes. Direct application of Dijkstra's algorithm on this discrete Cartesian graph remains a popular approximation method for this problem; however, the paths generated by such an approximation measure their cost metrics in a coordinate dependent manner,[‡] and are visibly segmented at the grid's resolution. In contrast, FMM approximations can generate paths with subgrid resolution (see section 3.1); paths that are reasonably smooth for practically sized grids. Furthermore, these approximations are theoretically convergent, meaning that the approximation approaches the true value function solution of (2) as $N \to \infty$ on all of the state space except a subset of measure zero.

We initialize the FMM by setting $V(x_s) = 0$ and $V(x_m) = \infty$ for all other nodes x_m (in practice we choose a large floating point value for ∞). We also

[‡]For example, Dijkstra on a square Cartesian grid measures distance with the Manhattan or 1-norm; in this norm the distance between two points depends on the alignment of the coordinate axes. While this axis alignment bias can be reduced by adding more edges to the graph, it will persist unless every possible path is enumerated by making the graph completely dense. The solution of the Eikonal equation (2) measures distance in the Euclidean or 2-norm, which is independent of axis alignment.

place x_s into a list ℓ. At each iteration of the FMM we remove the node x_m in ℓ with minimum value $V(x_m)$; this value is now fixed. We update $V(x_n)$ for each neighbor x_n of x_m with $V(x_n) > V(x_m)$. If any of those neighbors were not in ℓ already, we place them in ℓ. We then repeat, taking the node of next smallest value from ℓ and updating its neighbors, until no more nodes remain in ℓ. This procedure is the basis of Dijkstra's algorithm. Each node is removed from ℓ only once and has a constant number of neighbors. As we will see, updating each neighbor takes a constant amount of time. If a heap [17] or something similar is used to sort ℓ, the smallest node can be determined in logarithmic time, for a total cost $\mathcal{O}(dN^d \log N)$.

The only difference between Dijkstra's method and an FMM lies in the update equation for a node x_n [2]. Instead of considering each neighbor of x_n separately, we form a first order upwind finite difference approximation of $\nabla V(x_n)$ using various combinations of the neighbors x_p whose values $V(x_p)$ are less than the current approximation of $V(x_n)$. Plugging these finite difference approximations into (2) yields an implicit quadratic equation for the new approximation of $V(x_n)$. Detailed update formulas are given in the appendix.

To solve (4), we use an approximation scheme outlined in [3]. The "extension velocity" $F_{\text{ext}}(x)$ described in that paper is computed by solving

$$\nabla F_{\text{ext}}(x) \cdot \nabla V(x) = 0,$$

which is just (4) with a zero right hand side. In practice, we integrate the computation of $P_i(x)$ into the FMM computation of $V(x)$. When each node x_m is removed from ℓ, we compute $P_i(x_m)$ for each i. The first order upwind finite difference approximation of $\nabla V(x_m)$ is already known from the last update of $V(x_m)$, while $c_i(x_m)$ and $c(x_m)$ can be directly evaluated. Forming a first order upwind finite difference approximation of $\nabla P_i(x_m)$ using the same neighbor nodes that were used to build the approximation of $\nabla V(x_m)$ yields an implicit linear equation for $P_i(x_m)$. Note that the neighbors x_p of x_m involved with the approximation of $\nabla V(x_m)$ will all have $V(x_p) < V(x_m)$, so they will all have been removed from ℓ before x_m and hence will have known values $P_i(x_p)$. Again, detailed update formulas are given in the appendix.

3 Examples

For our example we consider planning a path for an aircraft flying across the idealized unit square country from lower left to upper right. The first cost function will be fuel, which we assume is a constant $c_{\text{fuel}}(x) = c_{\text{fuel}} = 1$. Because these are toy examples, we provide no specific units for our cost functions.

The second cost function will represent the threat of weather related problems $c_{\text{weather}}(x)$. Note that the intuitive quantification of weather threat would be the probability of encountering a storm along the flight path. This quantification cannot be used as a cost because probabilities are not additive; however, under an independence assumption they can be transformed into an additive cost by a logarithmic transformation. The figures and tables below assume that this transformation has been performed in generating $c_{\text{weather}}(x)$ from meteorologically determined storm probabilities.

Ideally, this weather forecast would be an accurate short term estimate of weather threat. When we examine a three cost example in section 3.2, we will assume that part of our fictional country is well monitored and can thus generate accurate short term weather threat estimates, while another part of the country is poorly monitored and in this region we are forced to resort to long term climatological estimates. Because these long term estimates are less accurate, we introduce a third cost function $c_{\text{uncertain}}(x)$ which will penalize paths through the poorly instrumented region of the country.

We focus on two dimensional examples primarily because three dimensional paths are very challenging to visualize on paper. While three dimensions is noticeably more expensive, we demonstrate in section 3.4 that it can still be done at interactive rates on the desktop.

The gradient descent procedure that generates the paths (explained in section 3.4) produces a series of waypoints leading from the source to the destination. In the plots that follow there is a small gap between the source location and the start of the paths. This gap appears because the source location is not explicitly added to the waypoint list; the gap is choosen small enough that the aircraft can fly a direct line between the source and the first waypoint (the beginning of the plotted path).

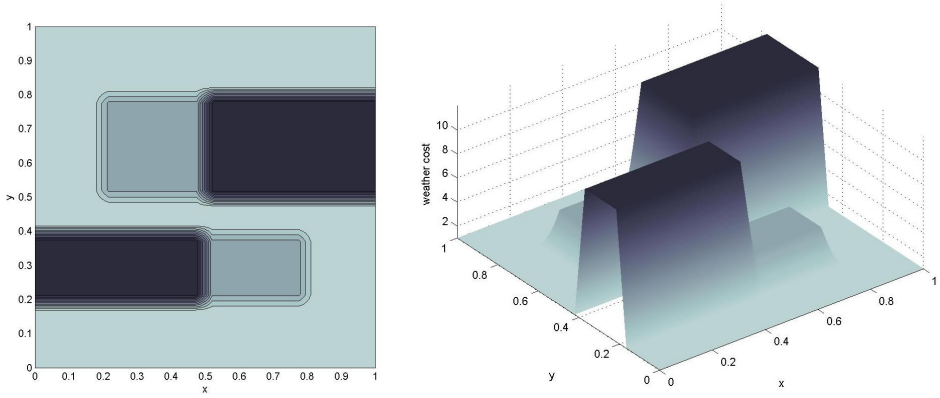

Figure 2: Weather threat cost function $c_{\text{weather}}(x)$

line type	minimize what cost?	fuel constraint	fuel cost	weather cost	λ
dotted	fuel	none	1.14	8.81	0.00
solid	weather	1.3	1.27	4.55	0.13
dashed	weather	1.6	1.58	3.03	0.62
dash dot	weather	none	2.69	2.71	1.00

Table 1: Properties of paths in figure 3.

3.1 Two Costs in Two Dimensions

Figure 2 shows a simple weather threat cost map $c_{\text{weather}}(x)$. Notice that the lower high threat bar extending from the left is slightly thinner than the upper high threat bar extending from the right.

Figure 3 shows four example paths plotted for various combinations of $c_{\text{weather}}(x)$ and c_{fuel} from the source $x_s = \begin{bmatrix}0.1 & 0.1\end{bmatrix}^T$ (marked by a star symbol) to the destination $x_d = \begin{bmatrix}0.9 & 0.9\end{bmatrix}^T$ (marked by a plus symbol). The combinations are described in table 1. In searching for paths that satisfy the fuel constraints, the range of λ was sampled uniformly. The λ values shown for the two constrained paths are the largest sampled λ for which the fuel constraint was satisfied. Notice in particular that the path under tight fuel constraints (the solid line) prefers to cross the thinner lower

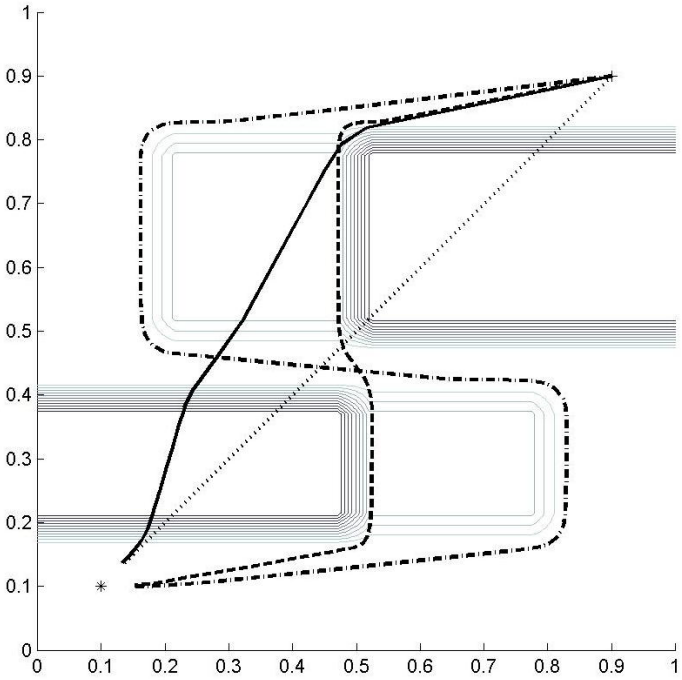

Figure 3: Some fuel and weather constrained paths. The properties of each path are explained in table 1.

bar of the weather cost function rather than the fuel symmetric path that exists crossing the thicker upper bar.

Figure 4 shows the points on the Pareto optimal curve of the destination location x_d generated by a uniform sampling of the space $\lambda \in [0, 1]$. Two explanations exist for those regions where the sample points are well spaced—either the uniform sampling was too coarse, or the Pareto curve is nonconvex. In the former case, a more intelligent sampling strategy could fill in the gaps inexpensively. Furthermore, even if the curve is nonconvex the existing samples provide fairly tight bounds on the degree of possible nonconvexity.

3.2 Three Costs in Two Dimensions

The first two cost functions are the same as in section 3.1: constant fuel $c_{\text{fuel}} = 1$ and $c_{\text{weather}}(x)$ from figure 2. For the third cost function, we

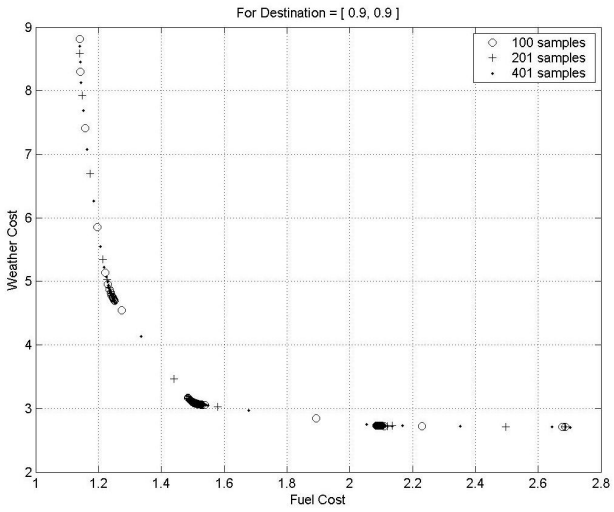

Figure 4: Sampling the Pareto optimal curve for a particular destination point.

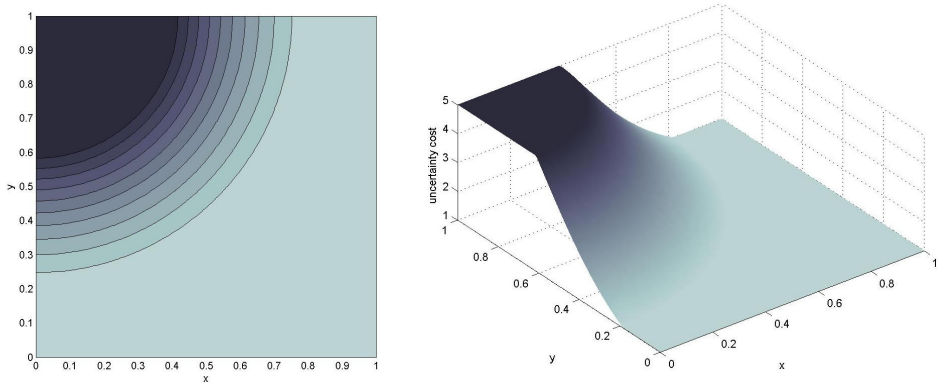

Figure 5: Uncertainty cost function $c_{\text{uncertain}}(x)$

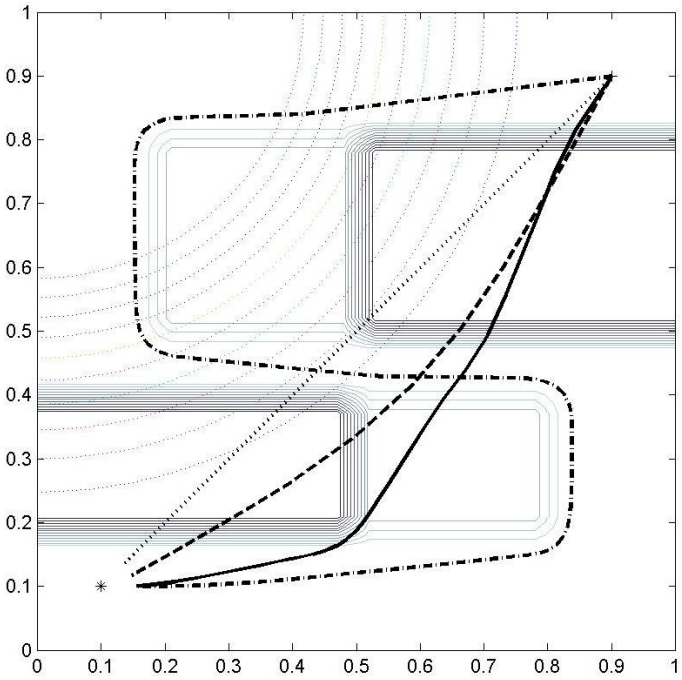

Figure 6: Some fuel, weather and uncertainty constrained paths. The properties of each path are explained in table 2.

assume that the upper left corner of our mythical country has few weather stations and therefore we create the uncertainty cost function $c_{\text{uncertain}}(x)$ shown in figure 5.

The resulting paths from x_s to x_d are shown in figure 6 and described in table 2. Because they optimize the same costs in the same manner, the dotted and dash dotted paths are basically the same as those shown in figure 3.[§] The most interesting path is that denoted by the solid line. Notice that the constraints on this path were satisfied by the tight fuel constrained path (also a solid line) in figure 3. In this case, however, we are penalizing paths that travel in the upper left portion of the map with the uncertainty cost $c_{\text{uncertain}}(x)$. Therefore, a path that crosses the thick high cost portion of the upper bar of the weather cost map is chosen; even though the weather

[§]The slight differences between their tabulated costs in tables 1 and 2 is due to the coarser state space grid used in this three constraint example.

line type	minimize what cost?	fuel constraint	weather constraint	fuel cost	weather cost	uncertainty cost
dotted	fuel	none	none	1.14	8.83	1.52
dash dot	weather	none	none	2.74	2.75	5.96
dashed	uncertainty	none	none	1.18	8.42	1.19
solid	uncertainty	1.3	6.0	1.25	5.85	1.25

Table 2: Properties of paths in figure 6.

cost is higher, it is still within the specified constraint and the resulting path's uncertainty cost is nearly as good as the minimum uncertainty cost path (given by the dashed line).

3.3 Two Costs in Three Dimensions

For a three dimensional problem, we plot a path from $x_s = \begin{bmatrix}0.1 & 0.1 & 0.1\end{bmatrix}^T$ to $x_d = \begin{bmatrix}0.9 & 0.9 & 0.9\end{bmatrix}^T$. The fuel cost function c_{fuel} remains a constant, while the weather cost $c_{\text{weather}}(x)$ has five stormy regions centered at the points:

$$\begin{bmatrix}0.1\\0.9\\0.9\end{bmatrix} \quad \begin{bmatrix}0.5\\0.5\\0.5\end{bmatrix} \quad \begin{bmatrix}0.9\\0.1\\0.1\end{bmatrix} \quad \begin{bmatrix}0.9\\0.5\\0.5\end{bmatrix} \quad \begin{bmatrix}0.1\\0.5\\0.5\end{bmatrix}$$

Each stormy region adds a scaled and shifted gaussian to $c_{\text{weather}}(x)$. In order to represent the general low level threat of unforeseen weather disturbances, we set $c_{\text{weather}}(x) = 1$ anywhere that the sum of the storm costs drops below unity. In order to break the symmetry of the problem, the first stormy region is 50% larger than the remaining four. A visualization of the weather cost function is shown in figure 7 along with three paths from x_s to x_d. The three volumetric shells in the figure represent (from faintest to darkest) the 1.1, 2.0 and 3.0 isosurfaces of $c_{\text{weather}}(x)$; its peak value is 5.5. The three demonstration paths are described in table 3.

3.4 The Implementation and Execution Times

To compute approximations of $V(x)$ and $P_i(x)$, we have implemented a version of the FMM described in section 2.4 in C++ for Cartesian grids.

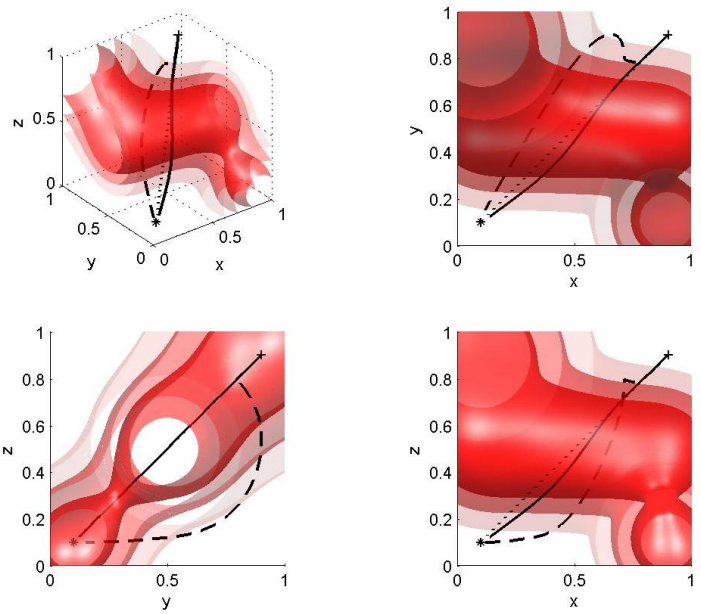

Figure 7: Some fuel and weather constrained paths in three dimensions. The properties of each path are explained in table 3.

line type	minimize what cost?	fuel constraint	fuel cost	weather cost
dotted	fuel	none	1.41	3.54
dashed	weather	none	1.64	1.64
solid	weather	1.55	1.55	2.00

Table 3: Properties of paths in figure 7.

section	d	k	N	$\Delta\lambda$	time (min)
3.1	2	2	201	0.005	0.5
3.2	2	3	101	0.020	1.0
3.3	3	2	101	0.010	22.3

Table 4: Run parameters for the examples (d = dimension, k = number of constraints, N = grid size, $\Delta\lambda$ = sample interval of convex combination cost). Time includes generation of the cost functions, PDE and ODE solves, and plotting all the figures.

While the code itself can handle any dimension, in practice the physical memory limits of desktop machines restrict the dimension to at most five even with very coarse grids. Using a MEX interface, these PDE solving routines can be called directly from Matlab.

To handle cost constrained paths, the sampling over λ is performed in Matlab. The speed of Matlab's interpreted language is not an issue in this case, because the inner loop of the λ sampling process is the compiled C++ but still relatively time consuming FMM algorithm. In all of the examples shown, the range of λ is sampled uniformly. If a particular destination point were known in advance, a directed sampling of λ could quickly yield more accurate results; for example, bisection in the two cost case. In fact, if a particular destination point is specified, the FMM can be run faster in some cases by applying a version of A* search on the list ℓ, rather than just selecting the node with minimum value ([10] discusses this technique in the discrete graph setting).

We have so far been generating a relatively small number of paths, so this process is handled with Matlab's extensive ODE integration facilities. Once a $\hat{\lambda}$ has been chosen such that any path constraints are satisfied, $V^{\hat{\lambda}}(x)$ is used in (3) to determine the path. Because $\nabla V^{\hat{\lambda}}$ may change direction significantly from one integration step to the next, (3) is a moderately stiff ODE. Consequently, we have found a variable stepsize, implicit trapezoidal integrator to be effective (Matlab's `ode23t`); however, other variable stepsize integrators—such as the high order explicit 4-5 Runge-Kutta—could also be used.

We summarize the parameters for the examples of the previous sections in table 4. The grid sizes and number of λ samples were chosen large enough to give decent results and not so large as to overburden the authors' desktop

N	time (s) per λ	ratio
51	0.01	
101	0.04	3.24
201	0.13	3.76
401	0.55	4.20
801	2.44	4.41

Table 5: Costs of refining the grid in two dimensions. Time per λ is the time to solve a single instance of the PDEs for $V(x)$ and two path integral costs $P_1(x)$ and $P_2(x)$. The ratio column shows the roughly quadratic growth in execution time as N is increased.

N	time (s) per λ	ratio
51	1.27	
101	12.66	9.99
201	125.46	9.91

Table 6: Costs of refining the grid in three dimensions. Time per λ is the time to solve a single instance of the PDEs for $V(x)$ and two path integral costs $P_1(x)$ and $P_2(x)$. The ratio column shows the slightly greater than cubic growth in execution time as N is increased.

computer. These timings and those below are for a 2 GHz, 1 GB Pentium 4 Dell Inspiron 8200 running Windows XP Professional, Matlab version 6.5 (release 13) and Matlab's lcc compiler.

Tables 5 and 6 demonstrate the costs of refining the PDE grid. Both tables assume only two path integral cost PDEs are solved; however, most of the time in the FMM algorithm is spent solving for $V(x)$, so the increase in time per λ sample of an additional path integral PDE is only 10%–20%. As mentioned previously, the asymptotic cost of the algorithm is $\mathcal{O}(dN^d \log N)$ per λ sample. The ratio columns of the two tables show the expected growth in execution time—slightly above quadratic with N for two dimension, slightly above cubic with N for three—in all but the coarsest two dimensional grids (where the roughly constant overhead of initialization will be relatively significant).

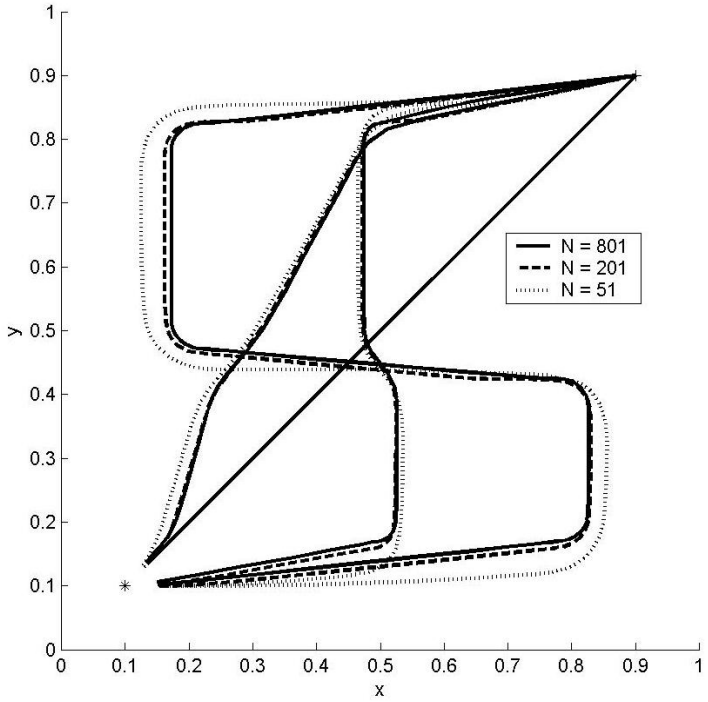

Figure 8: Comparing the path approximations generated by various grid resolutions. As the grid resolution improves, for almost every destination point the approximation converges to the analytically optimal path.

The time to solve (3) to generate a particular path is largely independent of the dimension or grid size, and is completely independent of the number of constraints or λ sampling interval. It will depend on the destination location (the closer to the source, the shorter the path). In our experience, most paths can be generated in less than a second, and few take more than three seconds. Using a compiled integrator (rather than Matlab's interpreted routines) would speed this process up even further.

To demonstrate the effects of refining or coarsening the grid on the quality of the resulting paths, figure 8 shows the paths from the example in section 3.1 generated for three different grid resolutions. The paths shown in figure 3 correspond to the $N = 201$ case. While grid refinement does yield visibly better paths, even the coarsest grid gets a qualitatively correct answer on even the most convoluted path.

4 Discussion

We have demonstrated an algorithm for constrained path planning in continuous state spaces for additive cost metrics and isotropic but inhomogenous and nonconvex cost functions. In those cases with multiple cost functions, a convex approximation of the Pareto optimal surface is explored; consequently, the algorithm may not find all feasible paths although in practice this has rarely been a problem. While the asymptotic cost of the algorithm is exponential in the dimension and in the number of cost functions (assuming uniform sampling of the Pareto optimal surface), it can be run at interactive rates on the desktop if their sum is five or less, and overnight if their sum is six.

There are several straightforward extensions of this work to more general path planning problems. We can immediately incorporate multiple source locations, by making each source a boundary condition with value zero of the PDEs (2) and (4). The resulting value function will generate paths from the nearest source to each destination state. Hard obstacles in the state space can be treated by either making the cost function very large in their interior or by making the obstacle's boundary a part of the PDEs' boundaries with very large value. Creating boundary nodes with intermediate values (neither zero nor very large) can be interpreted as penalizing those nodes as possible source locations. We can also swap the meaning of source and destination, in which case the value function can be used to generate a feedback control.

The basic FMM algorithm described in section 2.4 has been extended to unstructured meshes, and a more accurate second order approximation scheme has been developed. For more details on FMM and its extensions, we refer the reader to [18]. We are in the process of developing a version of FMM that runs on an adaptively refined Cartesian grid, so as to better represent problems with hard obstacles.

The current path planning formulation assumes that the cost of a path is a function only of its current state; this is equivalent to claiming that the vehicle which will execute the path can travel equally well in any direction from any location. This assumption is reasonable when the resolution of the grid is much coarser than the dynamics of the vehicle; for example, planning aircraft paths across a country. But on shorter time and space scales, it is unrealistic to assume that an airplane can make a sharp turn. Treating nontrivial vehicle dynamics requires that the cost functions

be anisotropic. Unfortunately, such anisotropy means that the value function will no longer be the viscosity solution of the eikonal equation (2), but rather a more general static Hamilton-Jacobi PDE. The FMM will not work on these PDEs; however, several algorithms have been proposed to solve them quickly [15, 19, 20, 21].

The additive path integral cost model used in this paper is very common, and includes multiplicative costs through a logarithmic transformation. Another common cost metric is maximum cost along the path. While maximum cost can currently be evaluated for a single path during the integration of (3), we are investigating methods capable of evaluating this metric over the entire state space. We are also examining efficient methods of approximating all of the Pareto optimal curve for each destination location, rather than just its convex hull.

Acknowledgements: We would like to thank Aniruddha Pant for suggesting the sweeping procedure over convex combinations of the multiple cost functions, and Professor Pravin Varaiya for the interpretation of this procedure as a convex approximation of the Pareto optimal surface and for several very useful discussions of value function properties. Thanks are also due to the Berkeley MICA team for providing the examples which originally motivated this work.

References

[1] J. N. Tsitsiklis, "Efficient algorithms for globally optimal trajectories," *IEEE Transactions on Automatic Control*, vol. AC-40, no. 9, pp. 1528–1538, 1995.

[2] J. A. Sethian, "A fast marching level set method for monotonically advancing fronts," *Proceedings of the National Academy of Sciences, USA*, vol. 93, no. 4, pp. 1591–1595, 1996.

[3] D. Adalsteinsson and J. A. Sethian, "The fast construction of extension velocities in level set methods," *Journal of Computational Physics*, vol. 148, pp. 2–22, 1999.

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

[5] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

[6] J. Barraquand, B. Langlois, and J. Latombe, "Numerical potential field techniques for robot path planning," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 22, no. 2, pp. 224–241, 1992.

[7] R. Kimmel and J. A. Sethian, "Optimal algorithm for shape from shading and path planning," *Journal of Mathematical Imaging and Vision*, vol. 14, no. 3, pp. 237–244, 2001.

[8] K. Konolige, "A gradient method for realtime robot control," in *International Conference on Intelligent Robots and Systems (IROS)*, vol. 1, (Takamatsu, Japan), pp. 639–646, 2000.

[9] A. Orda, "Routing with end-to-end QoS guarantees in broadband networks," *IEEE/ACM Transactions on Networking*, vol. 7, pp. 365–374, June 1999.

[10] G. Liu and K. G. Ramakrishnan, "A*prune: An algorithm for finding k shortest paths subject to multiple constraints," in *INFOCOM 2001*, vol. 2, pp. 743–749, 2001.

[11] A. Puri and S. Tripakis, "Algorithms for routing with multiple constraints," in *AIPS 2002 Workshop on Planning and Scheduling using Multiple Criteria*, (Toulouse, France), pp. 7–14, April 2002.

[12] A. Sei and W. W. Symes, "Convergent finite-difference traveltime gradient for tomography," in *Proceedings of* 65^{th} *Society of Exploration Geophysicists Annual Meeting*, (Houston, TX), pp. 1258–1261, 1995.

[13] M. G. Crandall, L. C. Evans, and P.-L. Lions, "Some properties of viscosity solutions of Hamilton-Jacobi equations," *Transactions of the American Mathematical Society*, vol. 282, no. 2, pp. 487–502, 1984.

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

[15] M. Falcone, "Numerical solution of dynamic programming equations," in *Optimal Control and Viscosity Solutions of Hamilton-Jacobi-Bellman equations*, Birkhäuser, 1997. Appendix A of [22].

[16] H.-K. Zhao, "Fast sweeping method for Eikonal equations I: Distance function," tech. rep., UCI, Department of Mathematics, University of California, Irvine, CA, 92697-3875, 2002. Under review, SINUM.

[17] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. San Franscisco: McGraw-Hill, 1990.

[18] J. A. Sethian, *Level Set Methods and Fast Marching Methods*. New York: Cambridge University Press, 1999.

[19] R. Kimmel and J. A. Sethian, "Computing geodesic paths on manifolds," *Proceedings of the National Academy of Sciences, USA*, vol. 95, no. 15, pp. 8431–8435, 1998.

[20] J. A. Sethian and A. Vladimirsky, "Ordered upwind methods for static Hamilton-Jacobi equations: Theory and algorithms," *SIAM Journal on Numerical Analysis*, vol. 41, no. 1, pp. 325–363, 2003.

[21] Y.-H. R. Tsai, L.-T. Cheng, S. Osher, and H.-K. Zhao, "Fast sweeping methods for a class of Hamilton-Jacobi equations," *SIAM Journal on Numerical Analysis*, vol. 41, no. 2, pp. 673–694, 2003.

[22] M. Bardi and I. Capuzzo-Dolcetta, *Optimal Control and Viscosity Solutions of Hamilton-Jacobi-Bellman equations*. Boston: Birkhäuser, 1997.

A Update Equations for the Fast Marching Method in Any Number of Dimensions

Section 2.4 described the basic FMM algorithm used to solve (2) for $V(x)$ and (4) for $P_i(x)$. In this appendix we give the update equations that form the heart of these algorithms: first for $V(x)$ and then for $P_i(x)$. These update equations are independent of dimension d, but work only on Cartesian grids. The update algorithm and equation for $V(x)$ given below is a version of those given in the appendix of [7], modified to treat grids with dimensionally dependent spacing (where h_j is the grid spacing in dimension j).

When a node x_m is removed from the list ℓ, any neighbor x_n with $V(x_n) > V(x_m)$ may need to be updated. Consider a specific neighbor node, which we label x_0. This node will have 2^d neighbors itself: one in each direction (we will call these directions left and right) in each dimension. Choose a

set of neighbor indices $\mathcal{I}$ by picking the neighbor (either left or right) with lowest value from each dimension (so $\mathcal{I}$ has d elements). If the grid spacing is equal in all dimensions, the nodes x_j for $j \in \mathcal{I}$ and the node x_0 are the vertices of a d dimensional simplex; otherwise they form a distorted simplex. The formula derived below calculates $V(x_0)$ as if the characteristic of (2) giving $V(x_0)$ its value came from this simplex.

It is possible that the characteristic in question flows along a lower dimensional face of the simplex rather than through its interior. Now we identify the subset of indices $\mathcal{J}$ from which the characteristic arises. First we define the following terms, where all of the summations are over the index set $\mathcal{J}$ (excepting the indices explicitly excluded).

$$
\begin{aligned}
T_1 &= \sum_j \left(\sum_{l \neq j} h_l^2 \right) V(x_j), \\
T_2 &= \sum_j \left(\sum_{l \neq j} h_l^2 \right) c^2(x_0), \\
T_3 &= \sum_{j_1} \sum_{j_2 \neq j_1} \left(\sum_{l \neq j_1, j_2} h_l^2 \right) \left[V(x_{j_1}) - V(x_{j_2}) \right]^2, \\
T_4 &= \sum_j \sum_{l \neq j} h_l^2.
\end{aligned}
$$

To find the appropriate $\mathcal{J}$, start with $\mathcal{J} = \mathcal{I}$. While

$$T_2 < T_3, \tag{5}$$

keep removing the node x_j with largest value $V(x_j)$ in $\mathcal{J}$. Once $T_2 \geq T_3$, use the remaining nodes in $\mathcal{J}$ to form first order upwind finite difference approximations of the partial derivatives of V at x_0, and plug these approximations into the square of (2) to get

$$\sum_{j \in \mathcal{J}} \left(\frac{V(x_j) - V(x_0)}{h_j} \right)^2 = c^2(x_0).$$

We can then use the quadratic equation to solve for $V(x_0)$.

$$\hat{V}(x_0) = \frac{T_1 + \left(\sum_j h_j \right) \sqrt{T_2 - T_3}}{T_4}. \tag{6}$$

The reader can verify that condition (5) ensures that the resulting discriminant in (6) is positive. If the resulting value $\hat{V}(x_0)$ is less than the existing value $V(x_0)$, then $\hat{V}(x_0)$ is taken as the new approximation of V at x_0.

Now consider the update of $P_i(x_m)$. Let $x_0 = x_m$ and remember the set $\mathcal{J}$ last used to update $V(x_0)$. Form first order upwind finite difference approximations for the partial derivatives of P_i and V at x_0, and plug these approximations into (4) to get

$$\sum_{j \in \mathcal{J}} \left(\frac{P_i(x_j) - P_i(x_0)}{h_j} \right) \left(\frac{V(x_j) - V(x_0)}{h_j} \right) = c_i(x_0) c(x_0).$$

Rearranging the terms yields the update equation (the sums are again over $\mathcal{J}$)

$$P_i(x_0) = \frac{\left(\sum_j \left[\sum_{l \neq j} h_l^2 \right] P_i(x_j) \left[V(x_j) - V(x_0) \right] \right) - c_i(x_0) c(x_0) \sum_j h_j^2}{\left(\sum_j \left[\sum_{l \neq j} h_l^2 \right] \left[V(x_j) - V(x_0) \right] \right)}.$$

Because this equation is solved only once for each x_0 and each P_i when that node x_0 is removed from list ℓ, computing the path integral cost functions is much cheaper per cost function than computing the value function.