Dynamic Programming Algorithms for Planning and Robotics in Continuous Domains and the Hamilton-Jacobi Equation

Ian Mitchell

Department of Computer Science

University of British Columbia

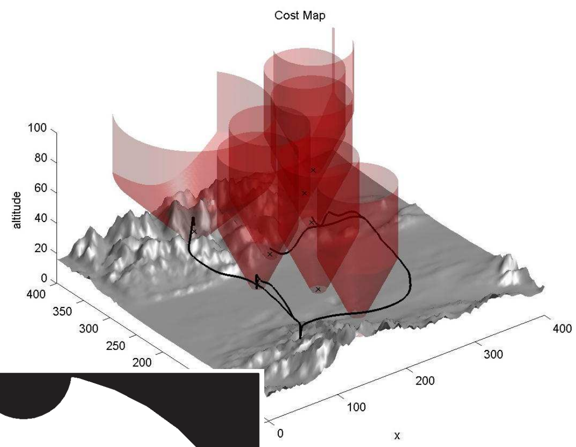
research supported by

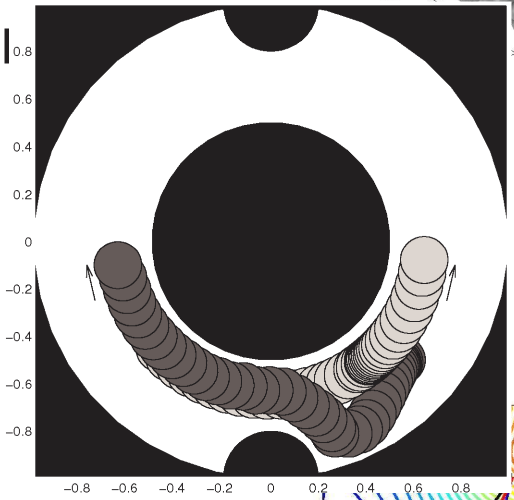
the Natural Science and Engineering Research Council of Canada

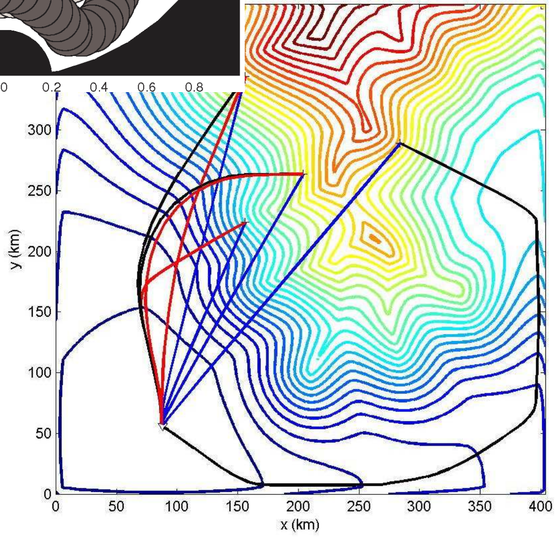
and Office of Naval Research under MURI contract N00014-02-1-0720

Outline

- Introduction
  - Optimal control
  - Dynamic programming (DP)
- Path Planning
  - Discrete planning as optimal control
  - Dijkstra's algorithm & its problems
  - Continuous DP & the Hamilton-Jacobi (HJ) PDE
  - The fast marching method (FMM): Dijkstra's for continuous spaces
- Algorithms for Static HJ PDEs
  - Four alternatives
  - FMM pros & cons
- Generalizations
  - Alternative action norms
  - Multiple objective planning

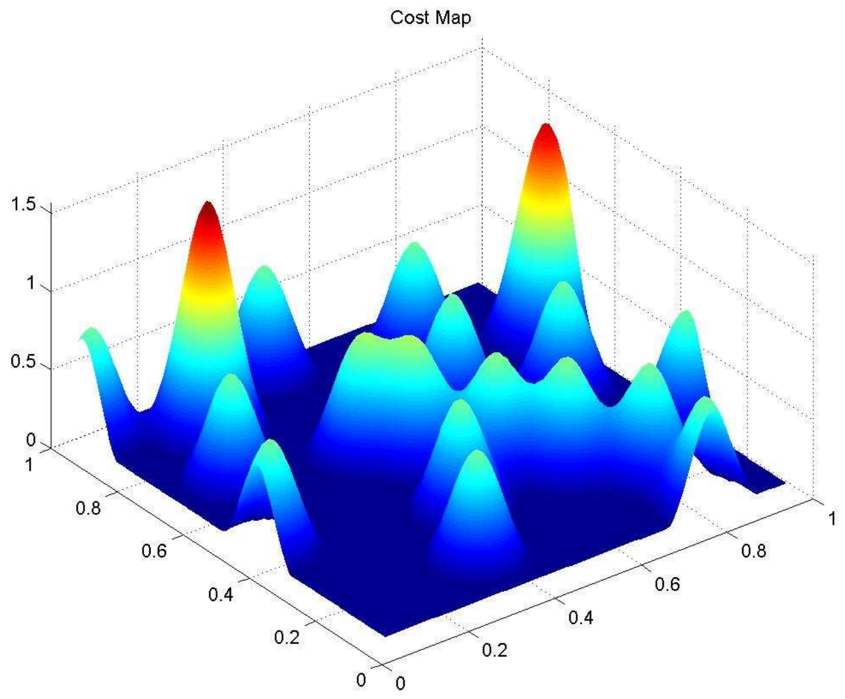




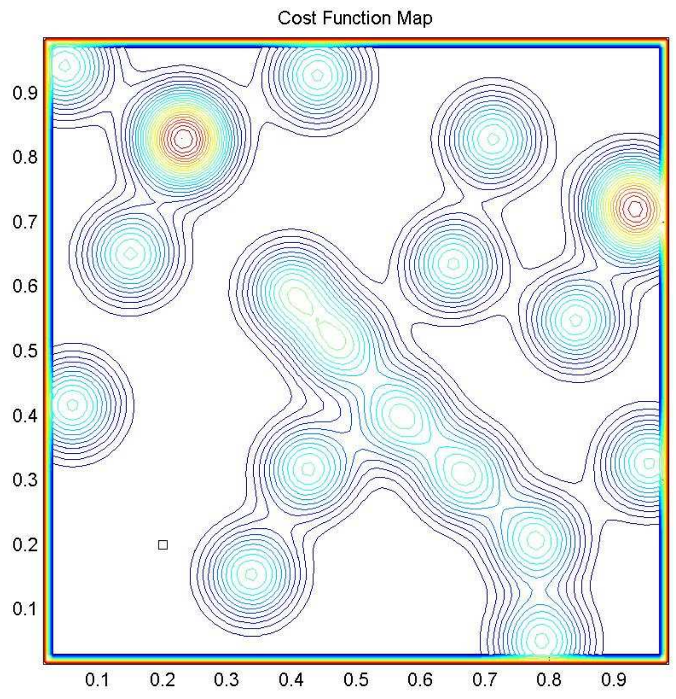


Basic Path Planning

- Find the optimal path $p(s)$ to a target (or from a source)
- Inputs
  - Cost $c(x)$ to pass through each state in the state space
  - Set of targets or sources (provides boundary conditions)



cost map (higher is more costly)



cost map (contours)

Discrete vs Continuous

- Discrete variable
  - Drawn from a countable domain, typically finite
  - Often no useful metric other than the discrete metric
  - Often no consistent ordering
  - Examples: names of students in this room, rooms in this building, natural numbers, grid of $\mathbb{R}^d$, …
- Continuous variable
  - Drawn from an uncountable domain, but may be bounded
  - Usually has a continuous metric
  - Often no consistent ordering
  - Examples: Real numbers [0, 1], $\mathbb{R}^d$, SO(3), …

Classes of Models for Dynamic Systems

- Discrete time and state
- Continuous time / discrete state
  - Discrete event systems
- Discrete time / continuous state
- Continuous time and state
- Markovian assumption
  - All information relevant to future evolution is captured in the state variable
  - Vital assumption, but failures are often treated as nondeterminism
- Deterministic assumption
  - Future evolution completely determined by initial conditions
  - Can be eased in many cases
- Not the only classes of models

$$x(t+1) = f(t, x(t))$$
where $x \in \{\ldots\} = \mathbb{S}$

$$x(t+1) = f(t, x(t))$$
where $x \in \mathbb{R}^d = \mathbb{S}$

$$\dot{x}(t) = f(t, x(t))$$
where $x(t) \in \mathbb{R}^d = \mathbb{S}$

Achieving Desired Behaviours

- We can attempt to control a system when there is a parameter u of the dynamics (the “control input”) which we can influence

$$\begin{aligned} x(t+1) &= f(x(t), u(t)) \text{ or} \\ \dot{x} &= f(x(t), u(t)) \\ &\text{for } x(t) \in \mathbb{S}, u(t) \in U(x(t)) \end{aligned}$$

  - Time dependent dynamics are possible, but we will mostly deal with time invariant systems

- Without a control signal specification, system is nondeterministic
  - Current state cannot predict unique future evolution
- Control signal may be specified
  - Open-loop $u(t)$ or u: $\mathbb{R} \to U$
  - Feedback, closed-loop $u(x(t))$ or u: $\mathbb{S} \to U$
  - Either choice makes the system deterministic again

Objective Function

- We distinguish quality of control by an objective / payoff / cost function, which comes in many different variations
  - eg: discrete time discounted with fixed finite horizon t_f

$$J(t_0, x_0, u(\cdot)) = \sum_{t=t_0}^{t_f - 1} \alpha^{t-t_0} g(x(t), u(t)) + \alpha^{t_f - t_0} g_f(x(t_f))$$

$$\text{where } x(\cdot) \text{ solves } x(t+1) = f(x(t), u(t)) \text{ and } x(t_0) = x_0$$

  - eg: continuous time no discount with target set T

$$J(x_0, u(\cdot)) = \int_{t_0}^{t_f} g(x(t), u(t)) + g_f(x(t_f))$$

$$\text{where } x(\cdot) \text{ solves } \dot{x}(t) = f(x(t), u(t)) \text{ and } x(t_0) = x_0$$

$$\text{and } t_f = \min\{t \mid x(t) \in T\}$$

Value Function

- Choose input signal to optimize the objective
  - Optimize: “cost” is usually minimized, “payoff” is usually maximized and “objective” may be either
- Value function is the optimal value of the objective function

$$
\begin{aligned}
V(t_0, x_0) &= \min_{u(\cdot)\in\mathcal{U}} J(t_0, x_0, u(\cdot)) \\
&= \min_{u(\cdot)\in\mathcal{U}} \sum_{t=t_0}^{t_f-1} \alpha^{t-t_0} g(x(t), u(t)) + \alpha^{t_f-t_0} g_f(x(t_f))
\end{aligned}
$$

or

$$
V(x_0) = \inf_{u(\cdot)\in\mathcal{U}} \int_{t_0}^{t_f} g(x(t), u(t)) + g_f(x(t_f))
$$

  - May not be achieved for any signal
  - Set of signals $\mathcal{U}$ can be an issue in continuous time problems (eg piecewise constant vs measurable)

Dynamic Programming in Discrete Time

- Consider finite horizon objective with $\alpha = 1$ (no discount)

$$\text{let } x(\cdot) \text{ solve } x(t+1) = f(x(t), u(t)) \text{ and } x(t_0) = x_0$$

$$\begin{aligned} J(t_0, x_0, u(\cdot)) &= \sum_{t=t_0}^{t_f-1} g(x(t), u(t)) + g_f(x(t_f)) \\ &= g(x(t_0), u(t_0)) + J(t_1, x(t_1), u(\cdot)) \\ &= g(x(t_0), u(t_0)) + J(t_1, f(x(t_1), u(t_1)), u(\cdot)) \end{aligned}$$

- So given $u(\cdot)$ we can solve inductively backwards in time for objective $J(t, x, u(\cdot))$, starting at $t = t_f$

$$J(t_f, x(t_f), u(\cdot)) = g_f(x(t_f))$$

$$J(t_i, x(t_i), u(\cdot)) = g(x(t_i), u(t_i)) + J(t_{i+1}, f(x(t_i), u(t_i)), u(\cdot))$$

  – Called dynamic programming (DP)

DP for the Value Function

- DP can also be applied to the value function
  - Second step works because $u(t_0)$ can be chosen independently of $u(t)$ for $t > t_0$

$$
\begin{aligned}
V(t_0, x_0) &= \min_{u(\cdot)} \sum_{t=t_0}^{t_f-1} g(x(t), u(t)) + g_f(x(t_f)) \\
&= \min_{u(\cdot)} \left(g(x(t_0), u(t_0)) + J(t_0 + 1, x(t_0 + 1), u(\cdot)) \right) \\
&= \min_{u} g(x(t_0), u) + V(t_0 + 1, x(t_0 + 1)) \\
&= \min_{u} g(x(t_0), u) + V(t_0 + 1, f(x(t_0), u(t_0))) \\
V(t_f, x(t_f)) &= g_f(x(t_f))
\end{aligned}
$$

Optimal Control via DP

- Optimal control signal

$$u^*(t, x(t)) = \operatorname*{argmin}_u \left[g(x(t), u) + V(t+1, f(x(t), u))\right]$$

- Optimal trajectory (discrete gradient descent)

$$\begin{aligned} x^*(t+1) &= f(x^*(t), u^*(t)) \\ &= \operatorname*{argmin}_{x(t+1)=f(x^*(t),u)} V(t+1, x(t+1)) \end{aligned}$$

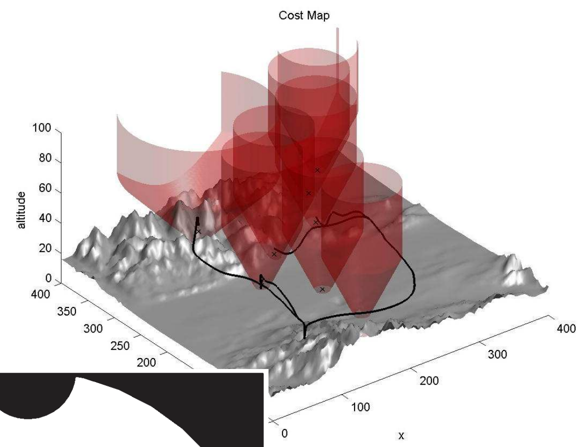
- Observe update equation

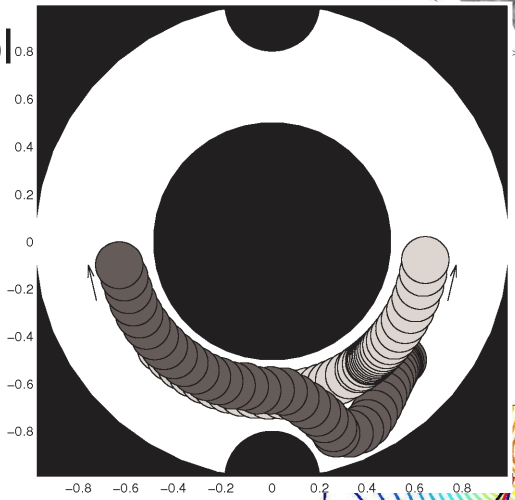
$$\begin{aligned} \Delta V(t, x) &= V(t+1, x(t+1)) - V(t, x) \\ &= -\min_u g(x, t, u) \end{aligned}$$

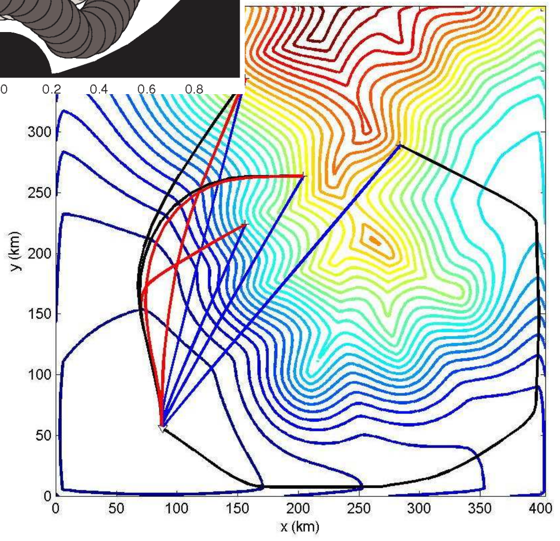
- Can be extended (with appropriate care) to
  - other objectives
  - probabilistic models
  - adversarial models

Outline

- Introduction
  - Optimal control
  - Dynamic programming (DP)
- Path Planning
  - Discrete planning as optimal control
  - Dijkstra's algorithm & its problems
  - Continuous DP & the Hamilton-Jacobi (HJ) PDE
  - The fast marching method (FMM): Dijkstra's for continuous spaces
- Algorithms for Static HJ PDEs
  - Four alternatives
  - FMM pros & cons
- Generalizations
  - Alternative action norms
  - Multiple objective planning

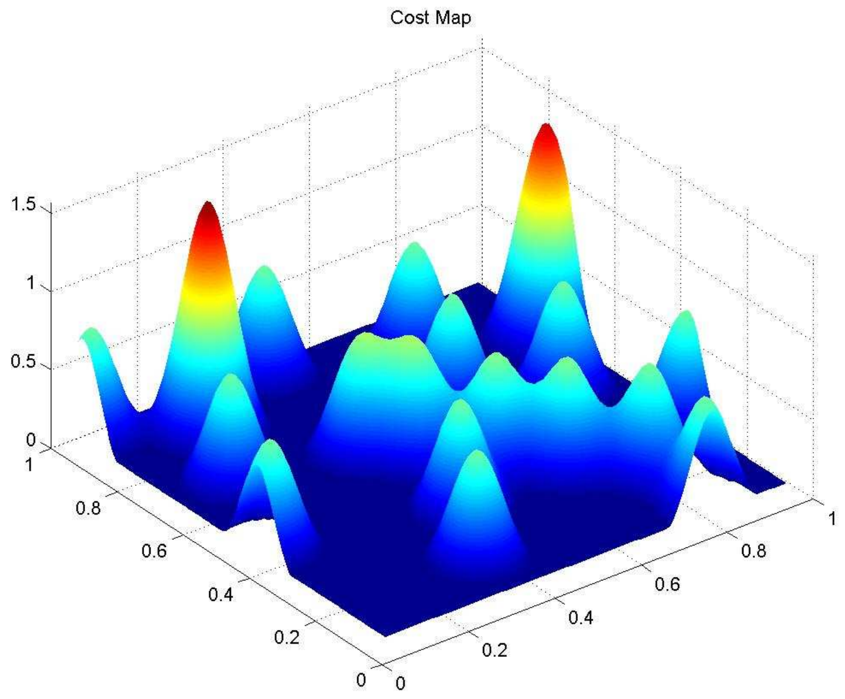




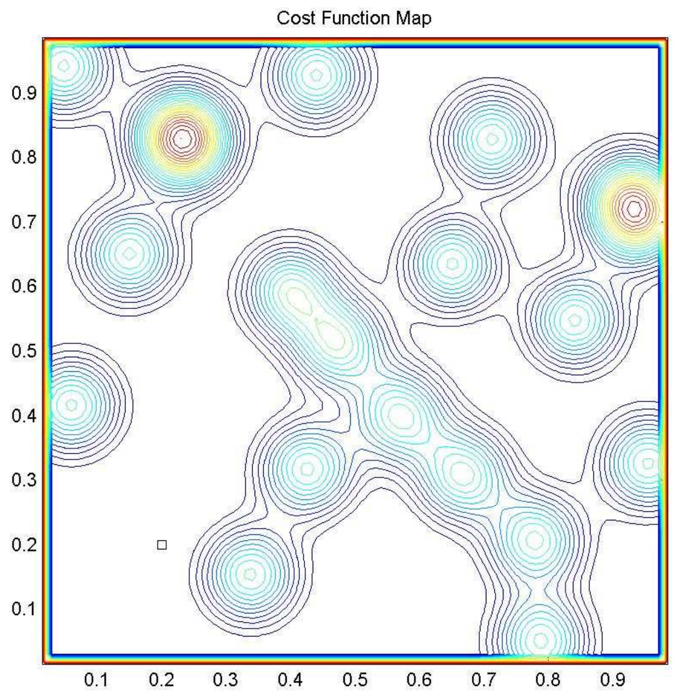


Basic Path Planning (reminder)

- Find the optimal path $p(s)$ to a target (or from a source)
- Inputs
  - Cost $c(x)$ to pass through each state in the state space
  - Set of targets or sources (provides boundary conditions)



cost map (higher is more costly)



cost map (contours)

Discrete Planning as Optimal Control

- Dynamics $x(t+1) = u$ where $u \in N(x(t))$
- Cost to reach target $\mathcal{T}$ is

$$J(x_0, u(\cdot)) = \sum_{t=0}^{T} \ell(x(t), u(t))$$

where

$$\ell(x, u) = c(x) \text{ and } T = \min\{t \geq 0 \mid x(t) \in \mathcal{T}\}$$

- Value function (min cost to target)

$$\vartheta(x_0) = \min_{u(\cdot)} J(x_0, u(\cdot))$$

- Value function solves recursion

$$\begin{aligned} \vartheta(x) &= \min_{y \in N(x)} [\vartheta(y) + c(x)] && \text{for } x \notin \mathcal{T} \\ \vartheta(x) &= 0 && \text{for } x \in \mathcal{T} \end{aligned}$$

Dynamic Programming Principle

$$\vartheta(x) = \min_{y \in N(x)} \left[\vartheta(y) + c(x)\right]$$

- Value function $\vartheta(x)$ is “cost to go” from x to the nearest target
- Value $\vartheta(x)$ at a point x is the minimum over all points y in the neighborhood $N(x)$ of the sum of
  - the value $\vartheta(y)$ at point y
  - the cost $c(x)$ to travel through x
- Dynamic programming applies if
  - Costs are additive
  - Subsets of feasible paths are themselves feasible
  - Concatenations of feasible paths are feasible
- Compute solution by value iteration
  - Repeatedly solve DP equation until solution stops changing
  - In many situations, smart ordering reduces number of iterations

Policy (Feedback Control)

- Given value function $\vartheta(x)$, optimal action at x is $x \to y$ where

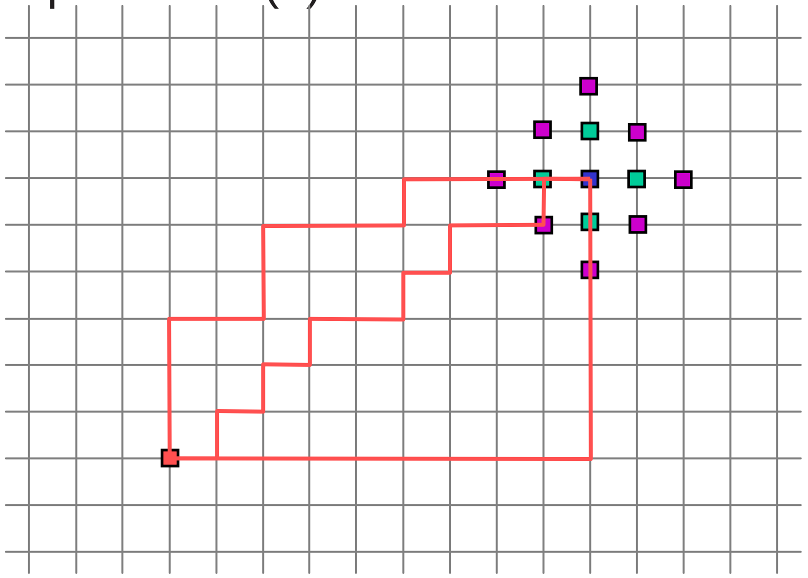
$$y = \text{argmin}\,\{y \in N(x) \mid \vartheta(y) + c(x)\}$$

  – Policy $u(x)$ = y

- Alternative policy iteration constructs policy directly
  – Finite termination of policy iteration can be proved for some situations where value iteration does not terminate
  – Representation of policy function may be more complicated than value function

Dijkstra's Algorithm for the Value Function

- Single pass dynamic programming value iteration on a discrete graph
  1. Set all interior nodes to a dummy value infinity ∞
  2. For all boundary nodes x and all $y \in N(x)$ approximate $\vartheta(y)$ by DPP
  3. Sort all interior nodes with finite values in a list
  4. Pop node x with minimum value from the list and update $\vartheta(y)$ by DPP for all $y \in N(x)$
  5. Repeat from (3) until all nodes have been popped



Constant cost map $c(y \to x) = 1$

🟦 Boundary node $\vartheta(x) = 0$

🟩 First Neighbors $\vartheta(x) = 1$

🟪 Second Neighbors $\vartheta(x) = 2$

🟥 Distant node $\vartheta(x) = 15$

Optimal path?

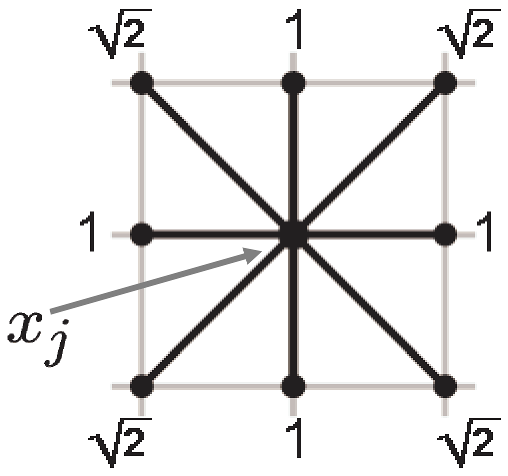
Generic Dijkstra-like Algorithm

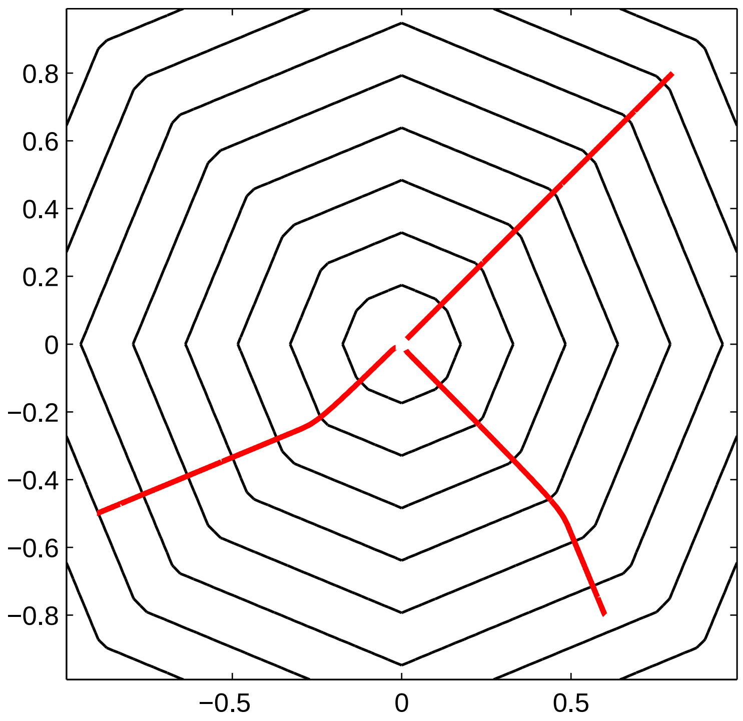
```
**foreach** $x_i \in \mathcal{G}_n \backslash \mathcal{T}$ **do** $\vartheta(x_i) \leftarrow +\infty$
**foreach** $x_i \in \mathcal{T}$ **do** $\vartheta(x_i) \leftarrow 0$
$\mathcal{Q} \leftarrow \mathcal{G}_n$
**while** $\mathcal{Q} \neq \emptyset$ **do**
    $x_i \leftarrow \text{ExtractMin}(\mathcal{Q})$
    **foreach** $x_j \in \mathcal{N}_n(x_i)$ **do**
        $\vartheta(x_j) \leftarrow \text{Update}(x_j, \mathcal{N}_n(x_j), \vartheta, c)$
```

- Could also use iterative scheme by minor modifications in management of the queue

Typical Discrete Update

- Much better results from discrete Dijkstra with eight neighbour stencil
- Result still shows facets in what should be circular contours



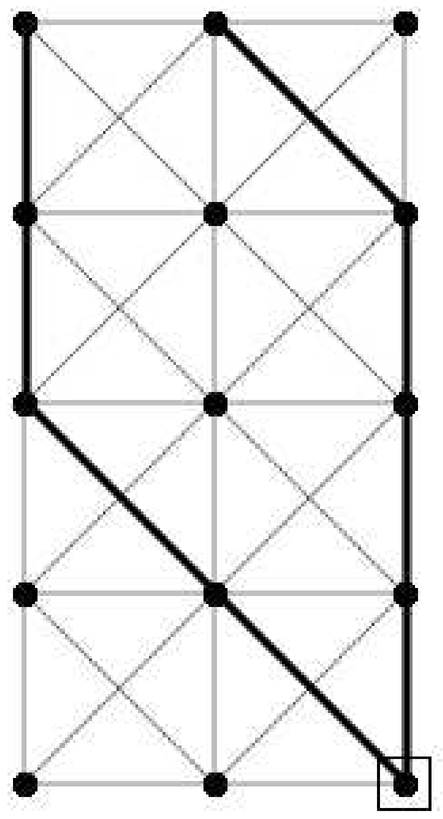


black: value function contours for minimum time to the origin
red: a few optimal paths

$$\text{Update}(x_j, N(x_j), V, c) = \min_{x_k \in N(x_j)} \left[c(x_j) + V(x_k) \right]$$

Other Issues

- Values and actions are not defined for states that are not nodes in the discrete graph
- Actions only include those corresponding to edges leading to neighboring states
- Interpolation of actions to points that are not grid nodes may not lead to actions optimal under continuous constraint


two optimal paths to the lower right node

Deriving Continuous DP (Informally)

- Discrete dynamic programming principle

$$\vartheta(x) = \min_{y \in N(x)} [\vartheta(y) + c(x \to y)]$$

- Continuous DPP for path $p(\cdot)$

$$\vartheta(p(s)) = \min_{p(\cdot)} \left[\vartheta(p(s + \Delta s)) + \int_s^{s+\Delta s} c(p(\sigma)) d\sigma \right]$$

- Rearrange

$$\min_{p(\cdot)} \left[\frac{\vartheta(p(s)) - \vartheta(p(s + \Delta s))}{\Delta s} - \frac{\int_s^{s+\Delta s} c(p(\sigma)) d\sigma}{\Delta s} \right] = 0$$

- Take limit $\Delta s \to 0$

$$\min_{p(\cdot)} \left[-\frac{d\vartheta(p(s))}{ds} - c(p(s)) \right] = 0$$

The Static Hamilton-Jacobi PDE

- After limit $\Delta s \to 0$

$$\min_{p(\cdot)} \left[\frac{d\vartheta(p(s))}{ds} + c(p(s)) \right] = 0$$

- Set $x = p(s)$ and apply chain rule

$$\min_{p(\cdot)} \left[\frac{\partial\vartheta(x)}{\partial x} \frac{dp(s)}{ds} + c(x) \right] = 0$$

- Let control be $u(s) = \frac{dp(s)}{ds}$, and observe that only dependence on $p(\cdot)$ is u

$$\min_u \left[D_x\vartheta(x) \cdot u + c(x) \right] = H(x, D_x\vartheta(x)) = 0$$

- From original problem, we get boundary conditions

$$\vartheta(x) = 0 \text{ for } x \in \partial\mathcal{T}$$

- Note: a very informal derivation

Continuous Planning as Optimal Control

- Dynamics $\dot{x} = u$, $x \in \mathbb{R}^2$ and $\|u\|_2 \leq 1$
- Cost to reach target is

$$J(x_0, u(\cdot)) = \int_0^T \ell(x(s), u(s))\, ds$$

  where

  $\ell(x, u) = c(x)$ and $T = \min\{t \geq 0 \mid x(t) \in \mathcal{T}\}$

- Value function (min cost to target)

$$\vartheta(x_0) = \inf_{u(\cdot)} J(x_0, u(\cdot))$$

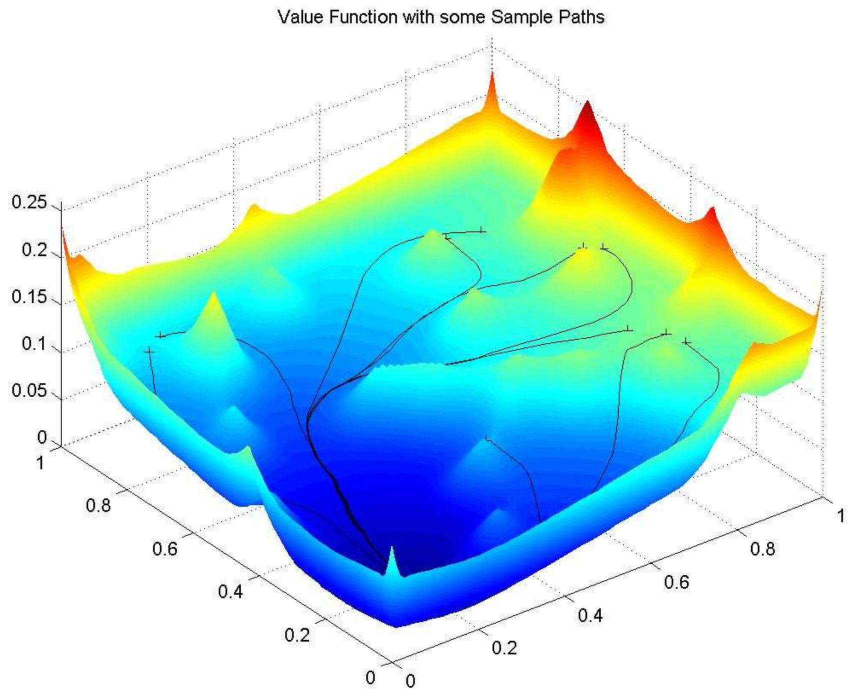
- Value function solves HJ PDE (we choose optimal control $u(\cdot) = -\frac{D_x\vartheta(x)}{\|D_x\vartheta(x)\|}$)

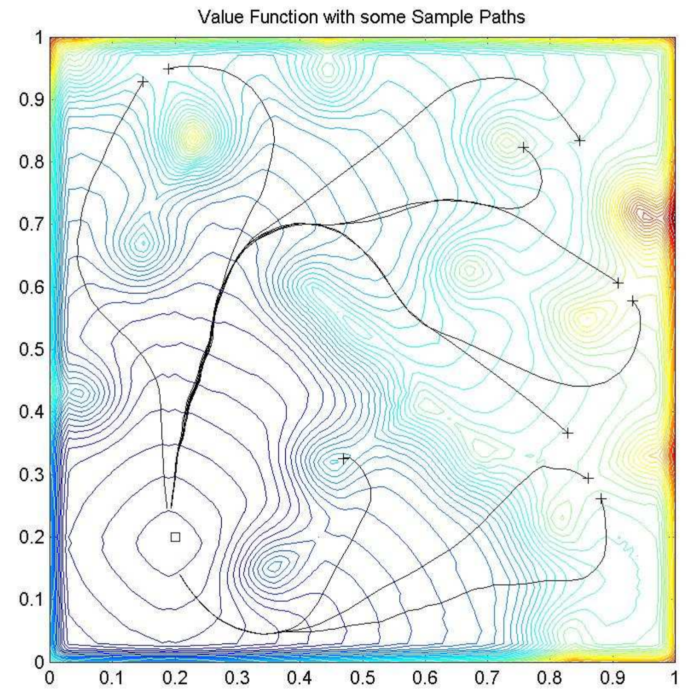
$$\begin{aligned} \|D_x\vartheta(x)\|_2 &= c(x) && \text{for } x \in \mathbb{R}^2 \setminus \mathcal{T} \\ \vartheta(x) &= 0 && \text{for } x \in \partial\mathcal{T} \end{aligned}$$

Path Generation

- Optimal path $p(s)$ is found by gradient descent
  - Value function $\vartheta(x)$ has no local minima, so paths will always terminate at a target

$$\frac{dp(s)}{ds} = u(s) = \frac{D_x\vartheta(x)}{\|D_x\vartheta(x)\|}$$

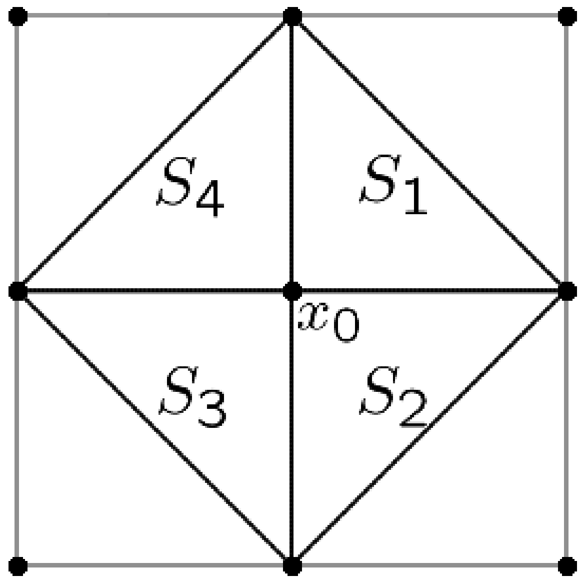




Allowing for Continuous Action Choice

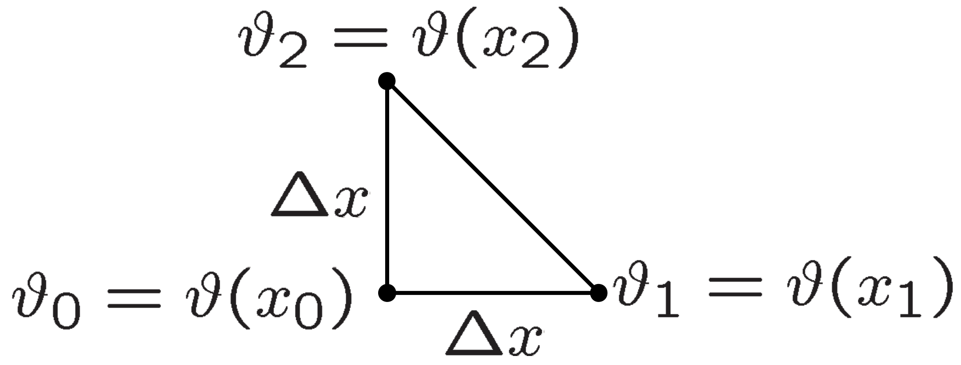
- Fast Marching Method (FMM): Dijkstra's algorithm adapted to a continuous state space
- Dijkstra's algorithm is used to determine the order in which nodes are visited
- When computing the update for a node, examine neighboring simplices instead of neighboring nodes
- Optimal path may cross faces or interior of any neighbor simplex

```
Input: $x_0, \mathcal{N}(x_0), \vartheta, c$
Output: $V(x_0)$
foreach $S \in \mathcal{N}_S(x_0)$ do
    Compute $\vartheta^{(S)}(x_0)$
return $\min_S \vartheta^{(S)}(x_0)$
```



Solution on a Simplex (Finite Difference)

We wish to solve
$\|D_x \vartheta(x)\|_2 = c(x)$


$\vartheta_2 = \vartheta(x_2)$
Δx
$\vartheta_0 = \vartheta(x_0)$
Δx
$\vartheta_1 = \vartheta(x_1)$

Construct finite difference approximation

$$\|D_x \vartheta(x_0)\|_2 = \sqrt{\left(\frac{\vartheta_0 - \vartheta_1}{\Delta x}\right)^2 + \left(\frac{\vartheta_0 - \vartheta_2}{\Delta x}\right)^2}$$

Then rearrange to find

$$\vartheta_0 = \frac{1}{2}\left(\vartheta_1 + \vartheta_2 + \sqrt{2\Delta x^2 c(x_0)^2 - (\vartheta_1 - \vartheta_2)^2}\right)$$

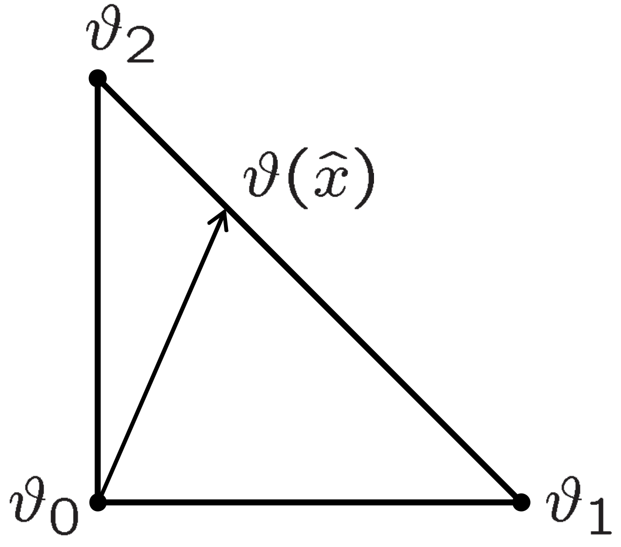
Solution on a Simplex (Semi-Lagrangian)

- We wish to find the optimal path across the simplex
- Approximate cost of travel across the simplex as constant $c(x_0)$
- Approximate cost to go from far edge of simplex as linear interpolation along the edge
- Optimization can be solved analytically; leads to the same solution as the finite difference approximation

$$\hat{x} = \lambda x_1 + (1-\lambda) x_2$$

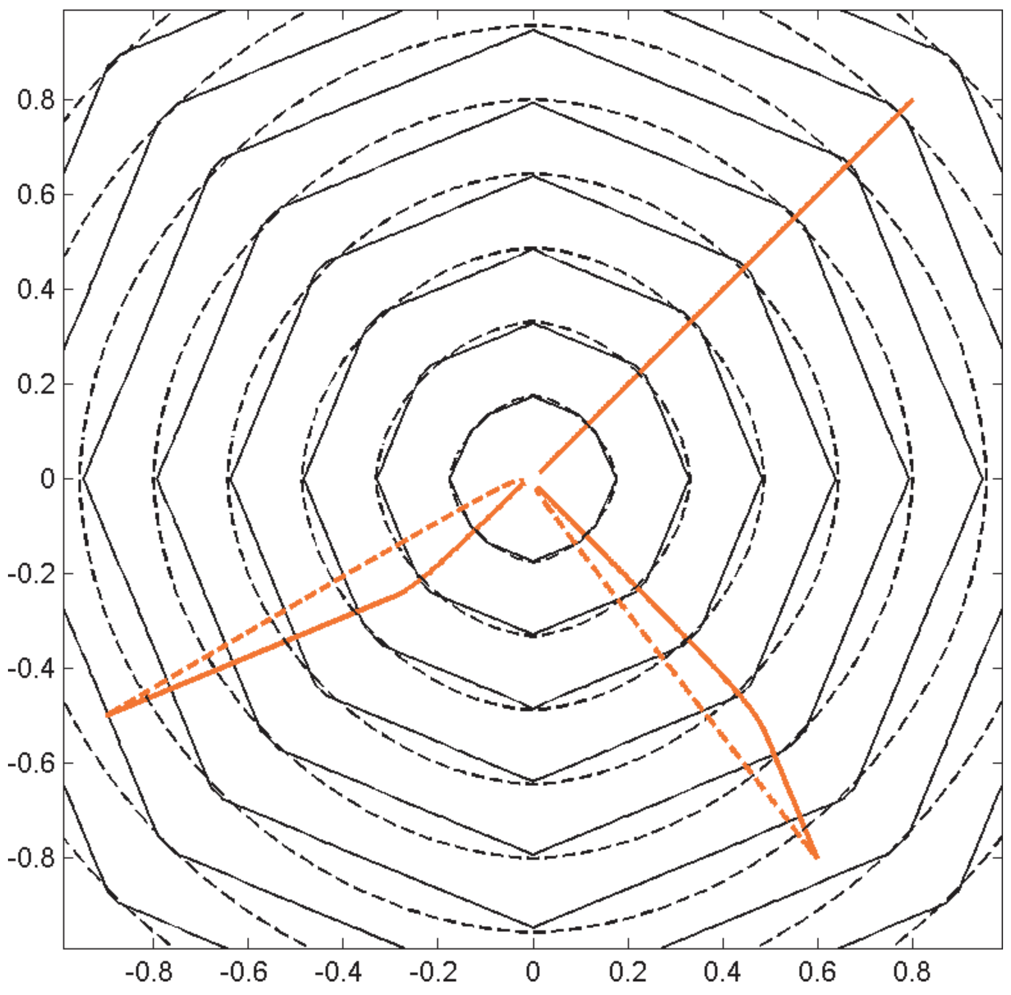
$$\vartheta(\hat{x}) = \lambda \vartheta_1 + (1-\lambda)\vartheta_2$$

$$\vartheta_0 = \min_{\lambda \in [0,1]} \vartheta(\hat{x}) + c(x)\|\hat{x} - x_0\|$$



How Do the Paths Compare?

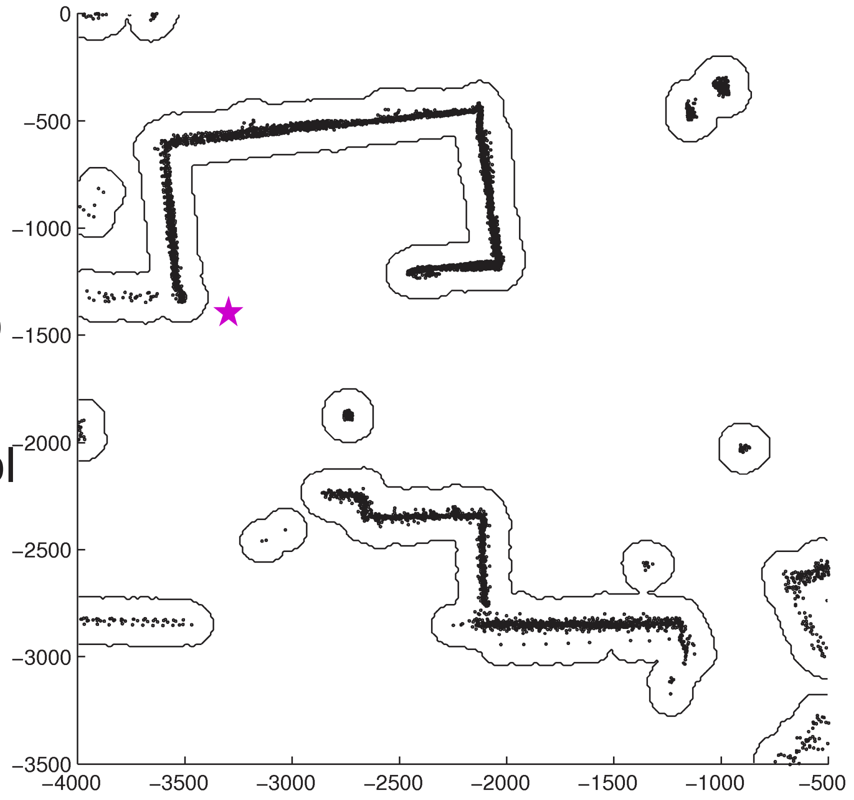
- Solid: eight neighbor discrete Dijkstra
- Dashed: Fast Marching on Cartesian grid



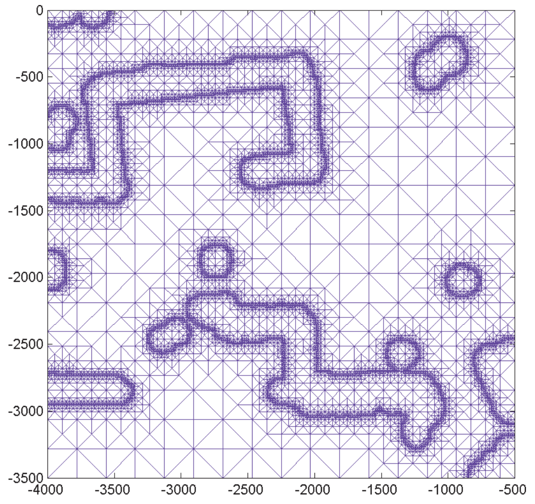
FMM for Robot Path Planning

- Find shortest path to objective while avoiding obstacles
  - Obstacle maps from laser scanner
  - Configuration space accounts for robot shape
  - Cost function essentially binary
- Value function measures cost to go
  - Solution of Eikonal equation
  - Gradient determines optimal control

typical laser scan with configuration space obstacles



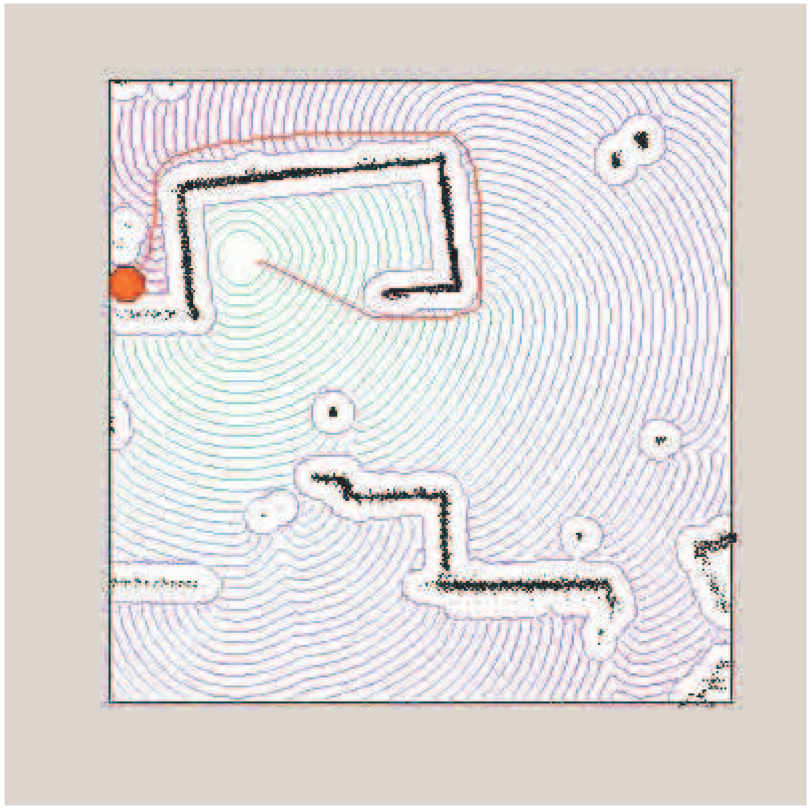
adaptive grid



Alton & Mitchell, "Optimal Path Planning under Different Norms in Continuous State Spaces," ICRA 2006

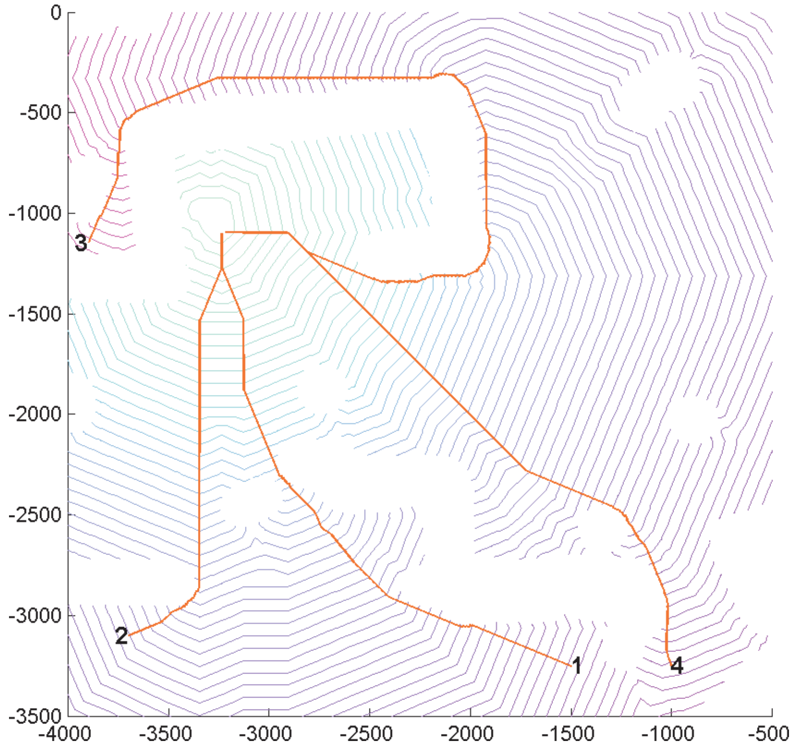
Continuous Value Function Approximation

- Contours are value function
  - Constant unit cost in free space, very high cost near obstacles
- Gradient descent to generate the path

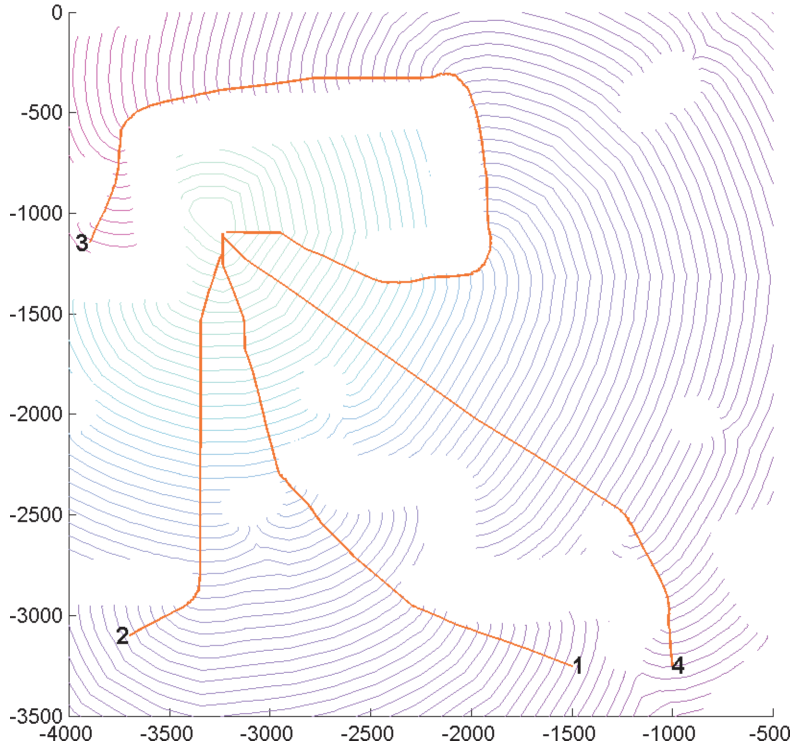


Comparing the Paths with Obstacles

- Value function from discrete Dijkstra shows faceting
  - Paths tend to follow graph edges even with action interpolation
- Value function from fast marching is smoother
  - Can still have large errors on large simplices or near target

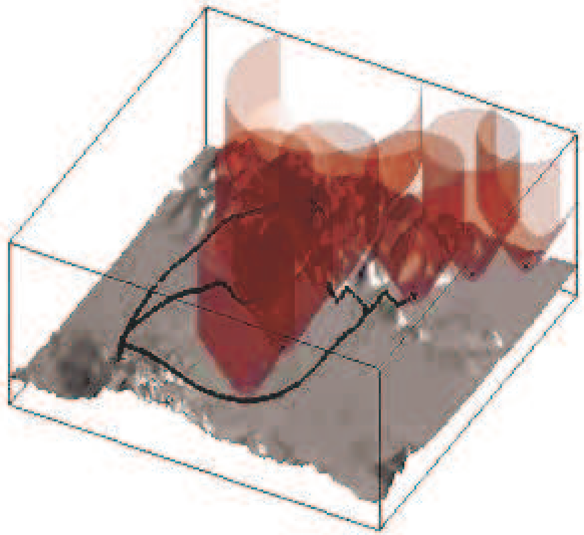


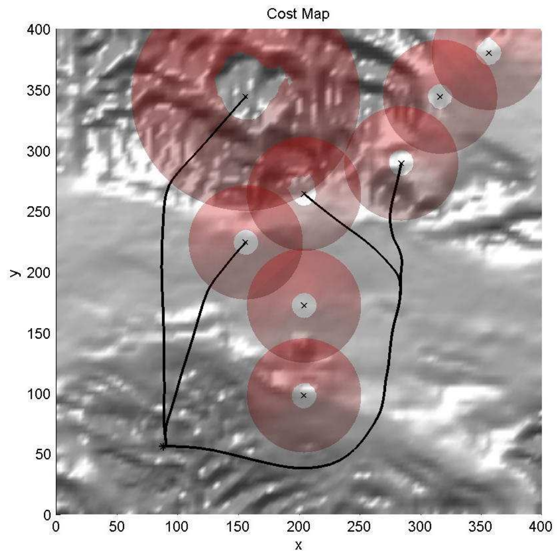
discrete Dijkstra's algorithm (8 neighbors)

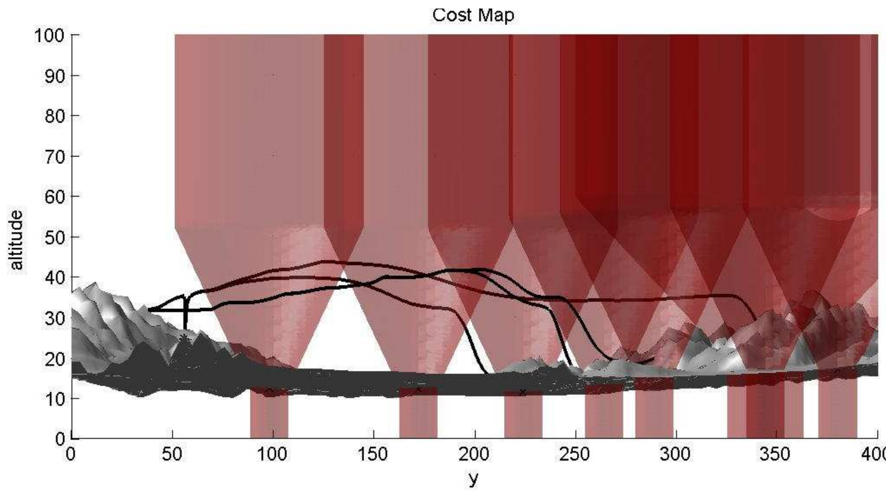


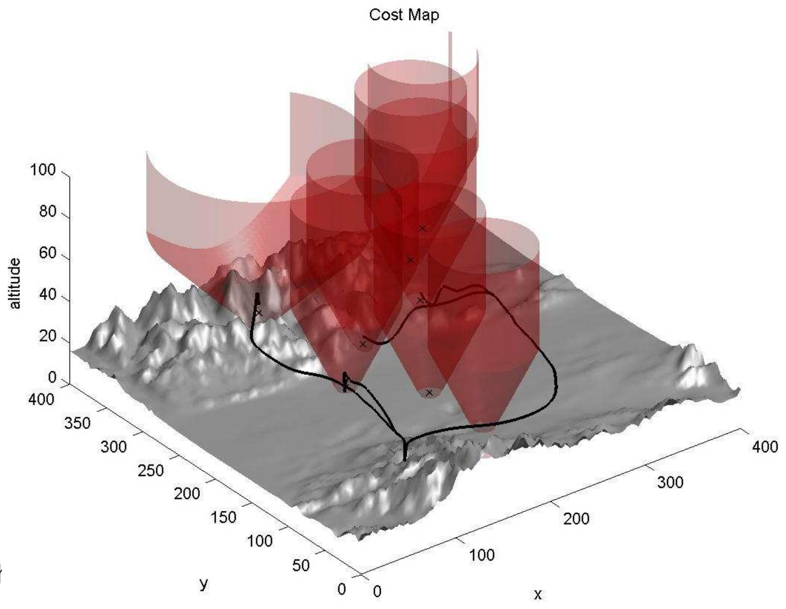
continuous fast marching method

Demanding Example? No!



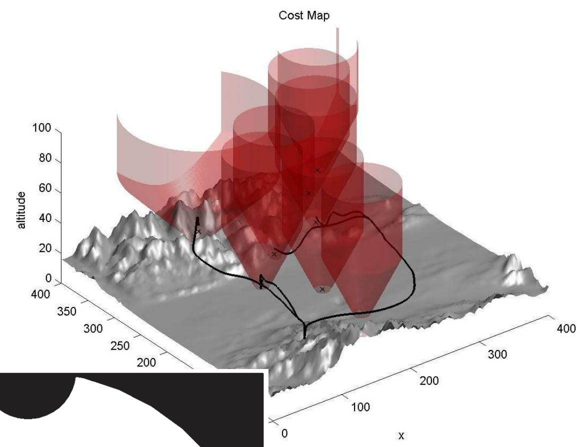


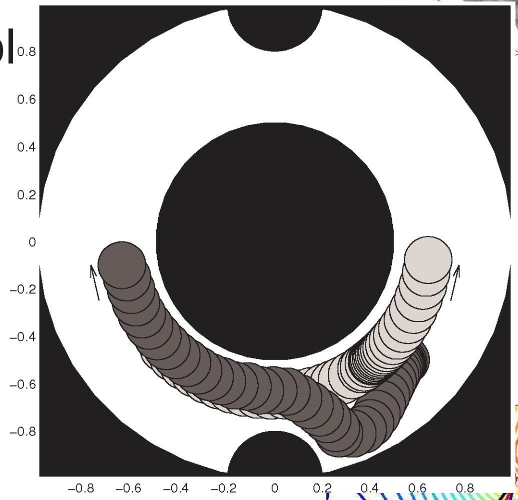


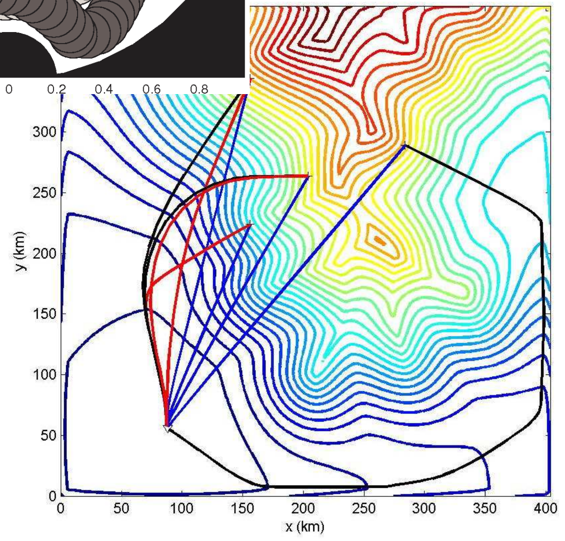


Outline

- Introduction
  - Optimal control
  - Dynamic programming (DP)
- Path Planning
  - Discrete planning as optimal control
  - Dijkstra’s algorithm & its problems
  - Continuous DP & the Hamilton-Jacobi (HJ) PDE
  - The fast marching method (FMM): Dijkstra’s for continuous spaces
- Algorithms for Static HJ PDEs
  - Four alternatives
  - FMM pros & cons
- Generalizations
  - Alternative action norms
  - Multiple objective planning







DP leads to Hamilton-Jacobi Equations

- Different cost functionals lead to different types of Hamilton-Jacobi equation
- Finite Horizon: fixed T

$$\phi(x(t), t) = \inf_{u(\cdot)} \left[\int_t^T \ell(x(s), u(s))\, ds + g(x(T)) \right]$$

solves for $x \in \mathbb{R}^n$ and $\phi(x, T) = g(x)$

$$D_t \phi(x, t) + \min_{u \in U} \left[f(x, u) \cdot D_x \phi(x, t) + \ell(x, u) \right] = 0$$

- Target Set: $T = \min\{t \geq 0 \mid x(t) \in \mathcal{T}\}$

$$\vartheta(x_0) = \inf_{u(\cdot)} \int_0^T \ell(x(s), u(s))\, ds$$

solves for $x \in \mathbb{R}^n \setminus \mathcal{T}$ and $\vartheta(\partial \mathcal{T}) = 0$

$$\min_{u \in U} \left[f(x, u) \cdot D_x \vartheta(x_0) + \ell(x, u) \right] = 0$$

Hamilton-Jacobi Flavours

- Time-dependent Hamilton-Jacobi used for dynamic implicit surfaces and finite horizon optimal control / differential games

$$D_t\phi(x,t) + H(x, D_x\phi(x,t)) = 0$$

  - Solution continuous but not necessarily differentiable
  - Time stepping approximation with high order accurate schemes
  - Numerical schemes have conservation law analogues
- Stationary (static) Hamilton-Jacobi used for target based cost to go and time to reach problems

$$H(x, D_x\vartheta(x)) = 0 \qquad \|D_x\vartheta(x)\| = c(x)$$

  - Solution may be discontinuous
  - Many competing algorithms, variety of speed & accuracy
  - Numerical schemes use characteristics (trajectories) of solution

Solving Static HJ PDEs

- Two methods available for using time-dependent techniques to solve the static problem
  - Iterate time-dependent version until Hamiltonian is zero
  - Transform into a front propagation problem
- Schemes designed specifically for static HJ PDEs are essentially continuous versions of value iteration from dynamic programming
  - Approximate the value at each node in terms of the values at its neighbours (in a consistent manner)
  - Details of this process define the “local update”
  - Eulerian schemes, plus a variety of semi-Lagrangian
- Result is a collection of coupled nonlinear equations for the values of all nodes in terms of all the other nodes
- Two value iteration methods for solving this collection of equations: marching and sweeping
  - Correspond to label setting and label correcting in graph algorithms

Convergence of Time-Dependent Version

$$H(x, D_x \vartheta(x)) = 0 \text{ for } x \in \Omega \setminus \mathcal{T}$$
$$\vartheta(x) = 0 \text{ for } x \in \partial\mathcal{T}$$

- Time-dependent version: replace $\vartheta(x) \to \vartheta(t,x)$ and add temporal derivative

$$D_t \vartheta(t, x) + H(x, D_x \vartheta(t, x)) = 0$$

  – Solve until $D_t\vartheta(t,x) = 0$, so that $\vartheta(t,x) = \vartheta(x)$
- Not a good idea
  – No reason to believe that $D_t\vartheta(t,x) \to 0$ in general
  – In limit $t \to \infty$, there is no guarantee that $\vartheta(t,x)$ remains continuous, so numerical methods may fail

Transformation to Time-Dependent HJ

Create implicit surface definition of $\mathcal{T}$

$$\varphi(x,0) \begin{cases} \leq 0, x \in \mathcal{T}; \\ = 0, x \in \partial\mathcal{T}; \\ \geq 0, x \in \mathbb{R}^d \setminus \mathcal{T}. \end{cases}$$

Under assumption $D_x\varphi(x,0) \cdot p \neq 0$ on $\partial\mathcal{T}$, make change of variables

$$D_x\vartheta(x) \leftarrow \frac{D_x\varphi(x,t)}{D_t\varphi(x,t)}$$

and get toolbox appropriate PDE

$$D_t\varphi(x,t) + \min_{p\in\mathbb{S}^1} \frac{D_x\varphi(x,t)\cdot p}{\ell(x,p)} = 0.$$

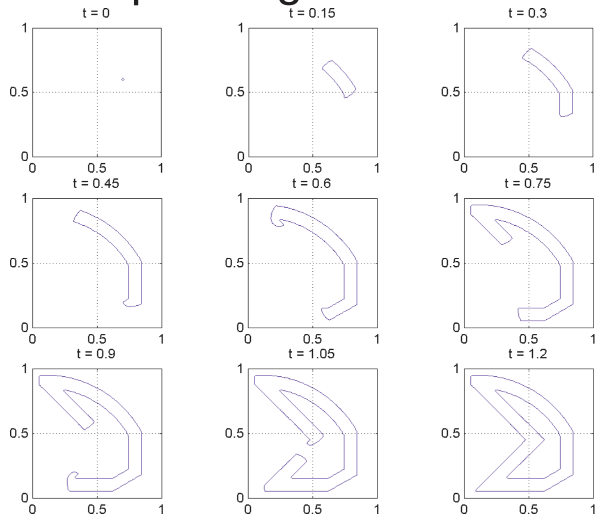
After solving, set ϑ to be crossing time

$$\vartheta(x) = \{t \mid \varphi(x,t) = 0\}.$$

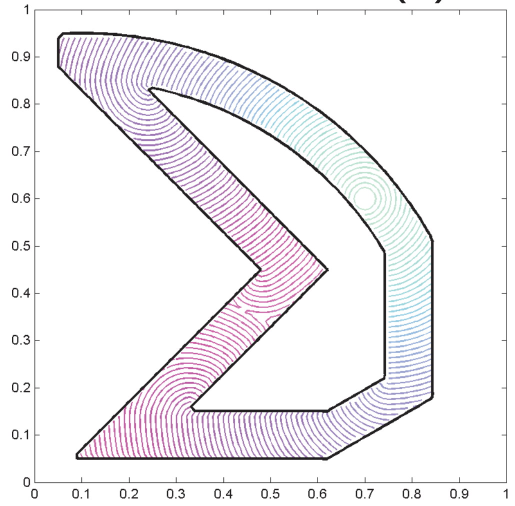
Methods: Time-Dependent Transform

- Equivalent to a wavefront propagation problem
- Pros:
  - Implicit surface function for wavefront is always continuous
  - Handles anisotropy, nonconvexity
  - High order accuracy schemes available on uniform Cartesian grid
  - Subgrid resolution of obstacles through implicit surface representation
  - Can be parallelized
  - ToolboxLS code is available (`http://www.cs.ubc.ca/~mitchell/ToolboxLS`)
- Cons:
  - CFL requires many timesteps
  - Computation over entire grid at each timestep

expanding wavefront

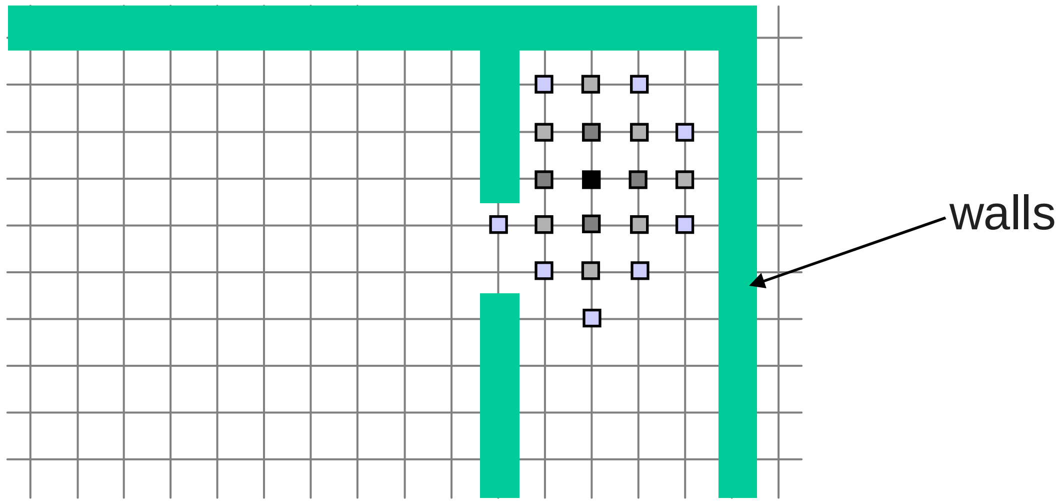


time to reach $\vartheta(x)$



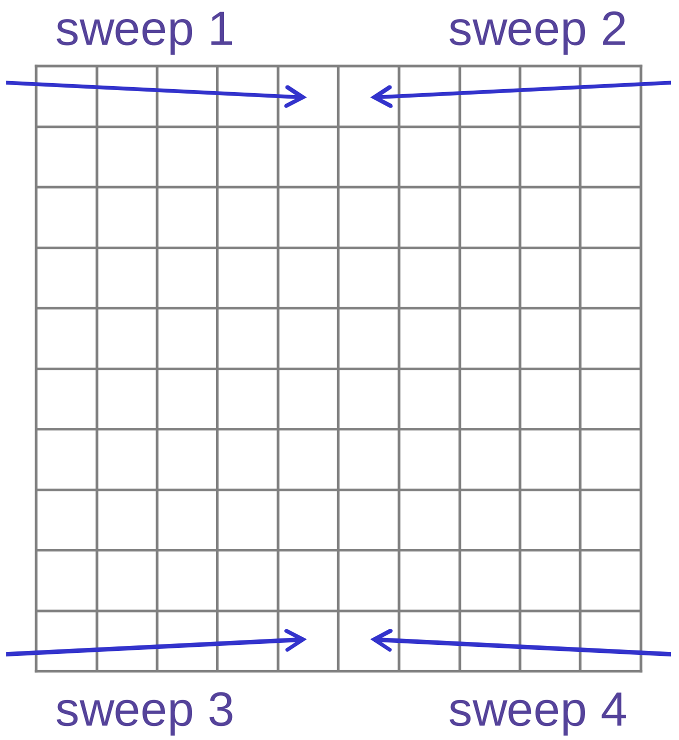
Methods: Fast Marching (FM)

- Dijkstra’s algorithm with a consistent node update formula
- Pros:
  - Efficient, single pass
  - Isotropic case relatively easy to implement
- Cons:
  - Random memory access pattern
  - No advantage from accurate initial guess
  - Requires causality relationship between node values
  - Anisotropic case (Ordered Upwind Method) trickier to implement



Methods: Fast Sweeping (FS)

- Gauss-Seidel iteration through the grid
  - For a particular node, use a consistent update (same as fast marching)
  - Several different node orderings are used in the hope of quickly propagating information along characteristics
- Pros:
  - Easy to implement
  - Predictable memory access pattern
  - Handles anisotropy, nonconvexity, obtuse unstructured grids
  - May benefit from accurate initial guess
- Cons:
  - Multiple sweeps required for convergence
  - Number of sweeps is problem dependent



Cost Depends on…

- So far assumed that cost depends only on position
- More generally, cost could depend on position and direction of motion (eg action / input)
  - Variable dependence on position: inhomogenous cost
  - Variable dependence on direction: anisotropic cost
- Discrete graph
  - Cost is associated with edges instead of nodes
  - Dijkstra’s algorithm is essentially unchanged
- Continuous space
  - Static HJ PDE no longer reduces to the Eikonal equation

$$\min_{u \in U} \left[D_x \vartheta(x) \cdot u + c(x) \right] = 0 \quad \not\Leftrightarrow \quad \| D_x \vartheta(x) \| = c(x)$$

$$\text{when } U \text{ is not a circle / sphere}$$

  - Gradient of ϑ may not be the optimal direction of motion

Interpreting Isotropic vs Anisotropic

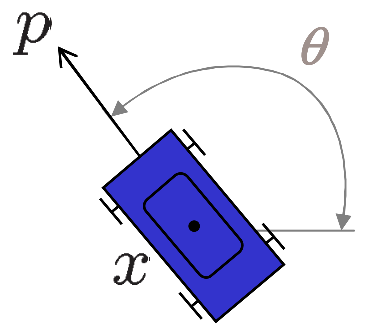
- For planar problems, cost can be interpreted as inverse of the speed of a robot at point x and heading θ = atan(p_2/p_1)
- General anisotropic cost depends on direction of motion

$$\dot{x} = \frac{p}{\ell(x,p)}$$

- Isotropic special case: robot moves in any direction with equal cost

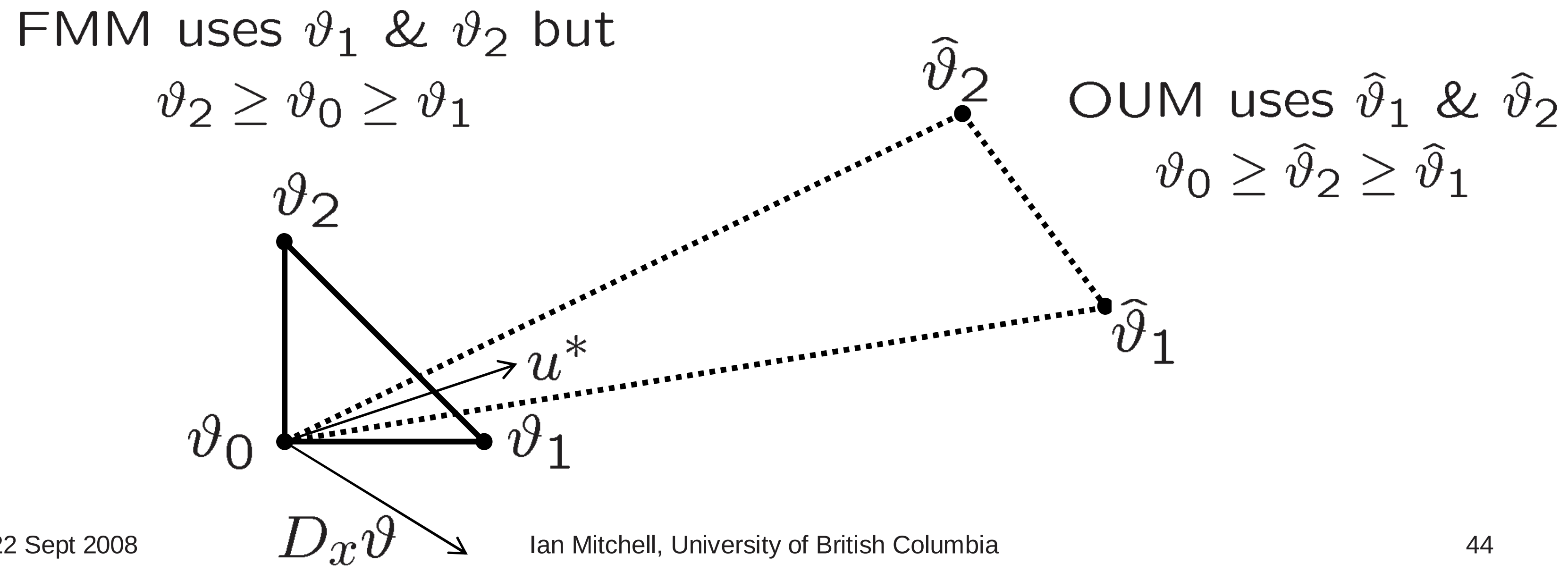
$$\dot{x} = \frac{p}{\ell(x)}$$

- Related to but a stronger condition than
  - holonomic
  - small time controllable



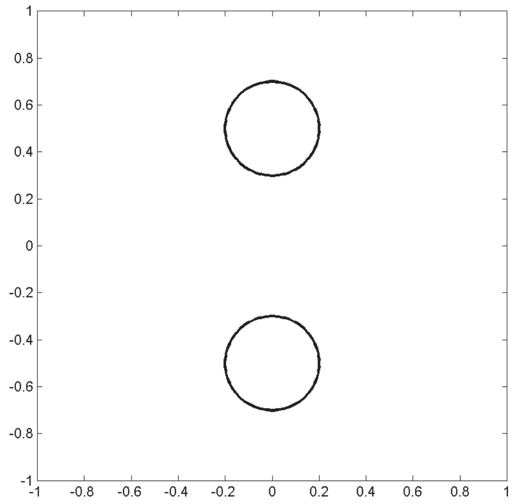
Anisotropy Leads to Causality Problems

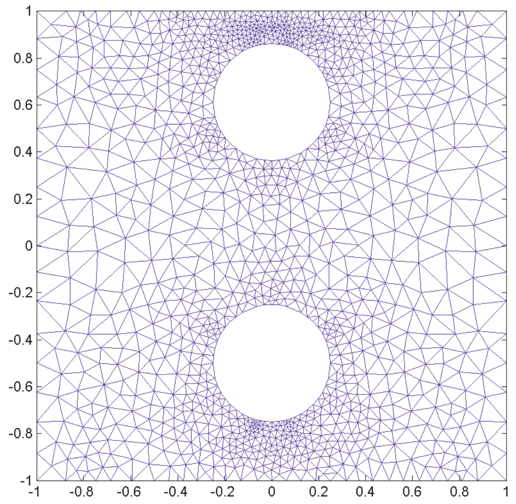
- To compute the value at a node, we look back along the optimal trajectory (“characteristic”), which may not be the gradient
- Nodes in the simplex containing the characteristic may have value greater than the current node
  - Under Dijkstra’s algorithm, only values less than the current node are known to be correct
- Ordered upwind (OUM) extension of FMM searches a larger set of simplices to find one whose values are all known

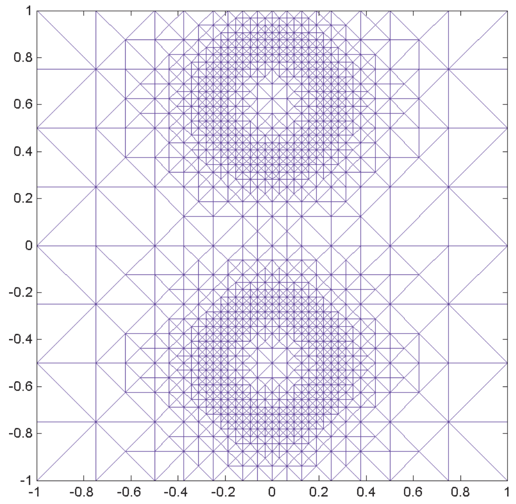

FMM uses ϑ_1 & ϑ_2 but
$\vartheta_2 \geq \vartheta_0 \geq \vartheta_1$
$\hat{\vartheta}_2$
OUM uses $\hat{\vartheta}_1$ & $\hat{\vartheta}_2$
$\vartheta_0 \geq \hat{\vartheta}_2 \geq \hat{\vartheta}_1$
ϑ_2
$\hat{\vartheta}_1$
u^*
ϑ_0
ϑ_1
$D_x\vartheta$

Representing Obstacles

- Computational domain should not include (hard) obstacles
  - Requires "body-fitted" and often non-acute grid: straightforward in 2D, challenging in 3D, open problem in 4D+
- Alternative is to give nodes inside the obstacle a very high cost
  - Side effect: the obstacle boundary is blurred by interpolation
- Improved resolution around obstacles is possible with semi-structured adaptive meshes
  - Not trivial in higher dimensions; acute meshes may not be possible

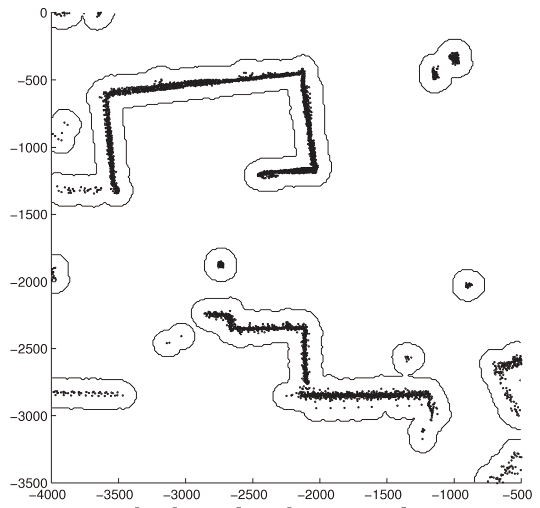

original obstacles

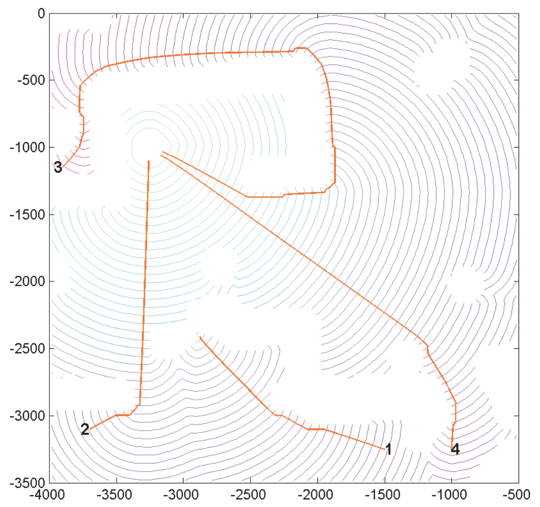

body fitted mesh

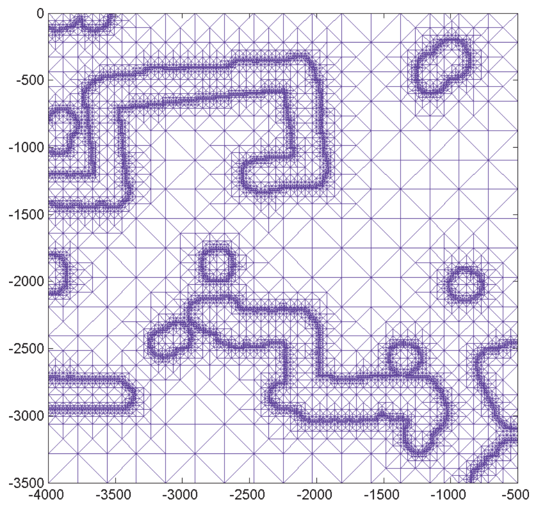

semi-structured mesh

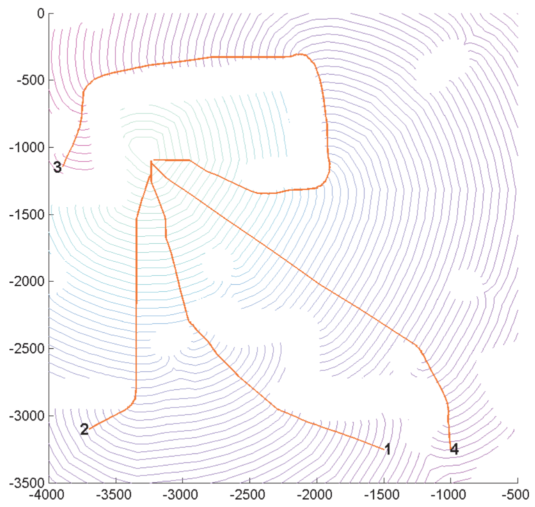
Adaptive Meshing is Practically Important

- Much of the static HJ literature involves only 2D and/or fixed Cartesian meshes with square aspect ratios
  - "Extension to variably spaced or unstructured meshes is straightforward…"
- Nontrivial path planning demands adaptive meshes
  - And C-space meshing, and dynamic meshing, and …


original obstacles


Cartesian mesh's paths


adaptive mesh


adaptive mesh's paths

FMM Does Not Do Nondeterminism

- Probabilistic
  - If stochastic behavior is Brownian, HJ PDE becomes (degenerate) elliptic (static HJ) or parabolic (time-dependent HJ)
  - Lots of theory available, but few algorithms
  - Leading error terms in approximation schemes often behave like dissipation / Brownian motion in dynamics
- Worst case / robust
  - Optimal control problem becomes a two player, zero sum differential game
  - Also called "robust optimal control"
  - Hamiltonian is not convex in $D_x\vartheta$ and causality condition may fail

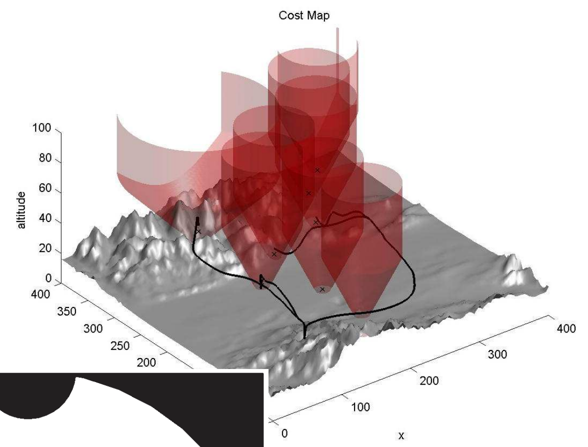
$$H(x, D_x\vartheta) = \max_{d\in D} \min_{u\in U} [D_x\vartheta(x) \cdot f(x,u,d) + c(x)]$$

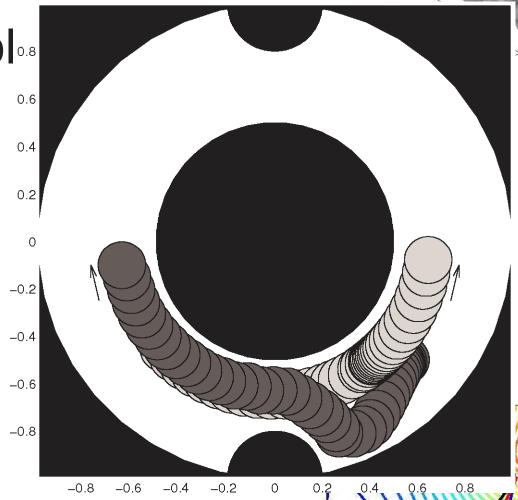
Other FMM Issues

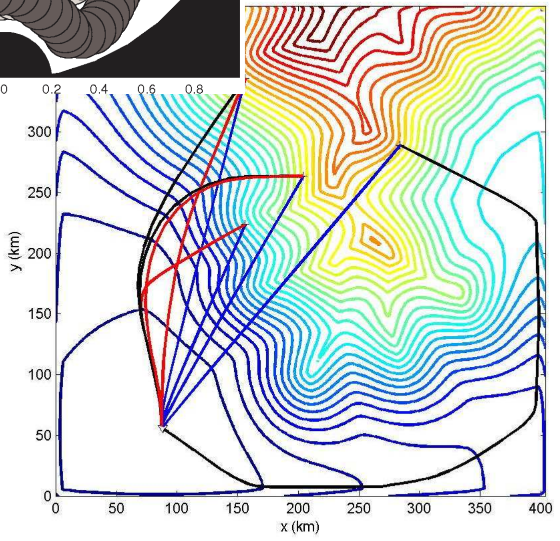
- Initial guess
  - FMM gets little benefit from a good initial guess because each node’s value is computed only when it might be completely correct
  - Changing the value of any node can potentially change any other node with a higher value, so an efficient updating algorithm is not trivial to design
- Focused algorithms (when given source and destination)
  - A* is a version of Dijkstra’s algorithm that ignores some nodes which cannot be on the optimal path
  - FMM updates depend on neighboring simplices rather than individual nodes, so there is no straightforward adaptation of A*
- Non-holonomic
  - The value function may not be continuous if some directions of motion are forbidden
  - Without continuity on a simplex, interpolation should not be used in the local updates

Outline

- Introduction
  - Optimal control
  - Dynamic programming (DP)
- Path Planning
  - Discrete planning as optimal control
  - Dijkstra's algorithm & its problems
  - Continuous DP & the Hamilton-Jacobi (HJ) PDE
  - The fast marching method (FMM): Dijkstra's for continuous spaces
- Algorithms for Static HJ PDEs
  - Four alternatives
  - FMM pros & cons
- Generalizations
  - Alternative action norms
  - Multiple objective planning



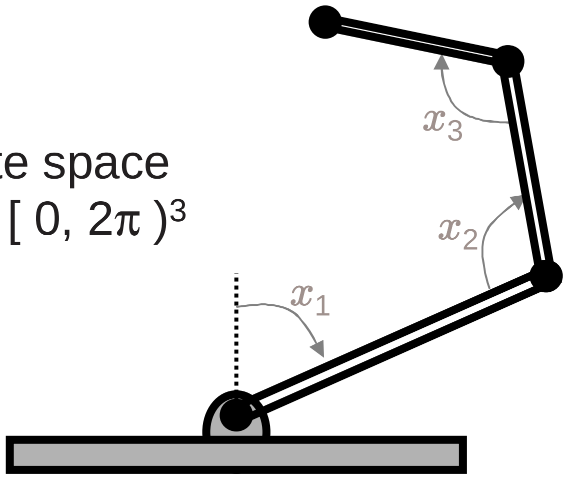




Why the Euclidean Norm?

- We have thus far assumed $\|\cdot\|_2$ bound, but it is not always best
- For example: robot arm with joint angle state space
  - All joints may move independently at maximum speed: $\|\cdot\|_\infty$
  - Total power drawn by all joints is bounded: $\|\cdot\|_1$
- If action is bounded in $\|\cdot\|_p$, then value function is solution of "Eikonal" equation $\|\vartheta(x)\|_{p^*} = c(x)$ in the dual norm p^*
  - $p = 1$ and $p = \infty$ are duals, and $p = 2$ is its own dual
- Straightforward to derive update equations for $p = 1$, $p = \infty$

state space
$x \in [\, 0, 2\pi\,)^3$



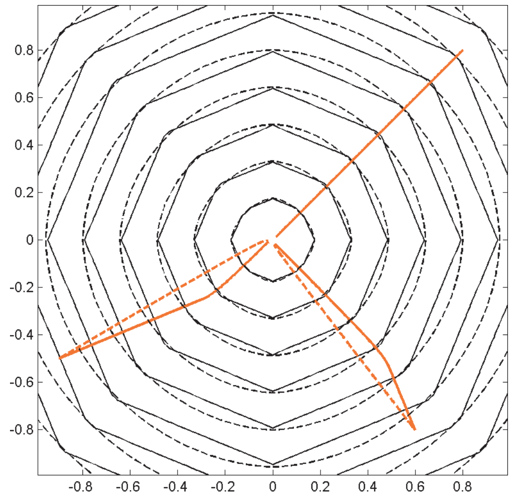
Alton & Mitchell
ICRA 2006
and
accepted to
SINUM 2008

Update Formulas for Other Norms

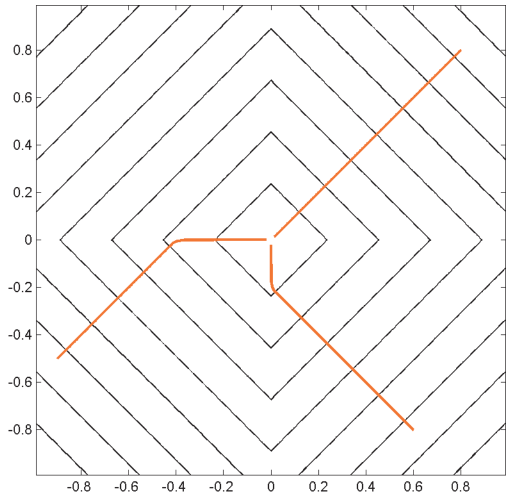
- Straightforward to derive update equations for $p = 1$, $p = \infty$

$$\|\nabla\vartheta(x_0)\|_1 = \frac{1}{\Delta x}\left(|\vartheta_0 - \vartheta_1| + |\vartheta_0 - \vartheta_2|\right)$$
$$\vartheta_0|_{p*=1} = \tfrac{1}{2}\left(\Delta x c(x_0) + \vartheta_1 + \vartheta_2\right)$$

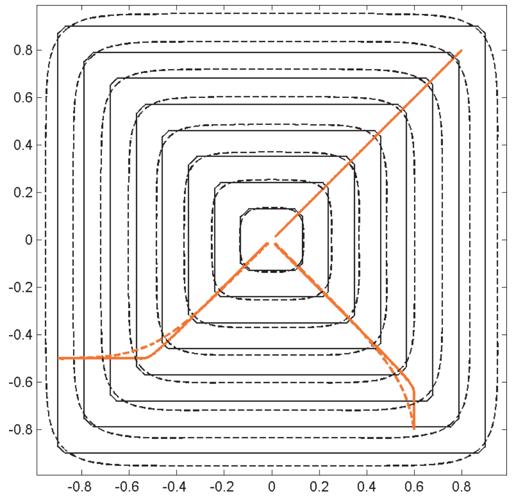
$$\|\nabla\vartheta(x_0)\|_\infty = \frac{1}{\Delta x}\max\left(|\vartheta_0 - \vartheta_1|, |\vartheta_0 - \vartheta_2|\right)$$
$$\vartheta_0|_{p*=\infty} = \Delta x c(x_0) + \min\left(\vartheta_1, \vartheta_2\right)$$



$p* = 2$



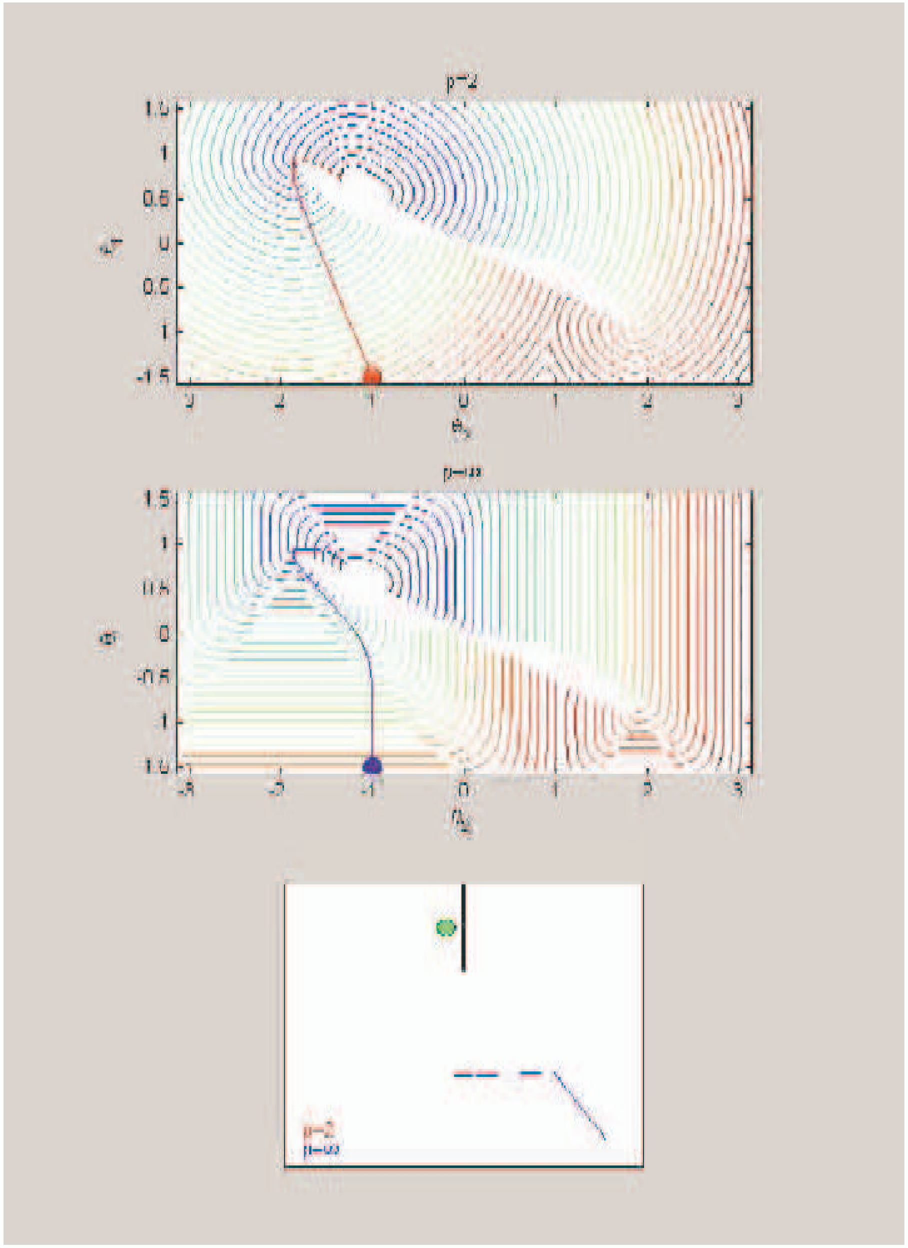
$p* = \infty$

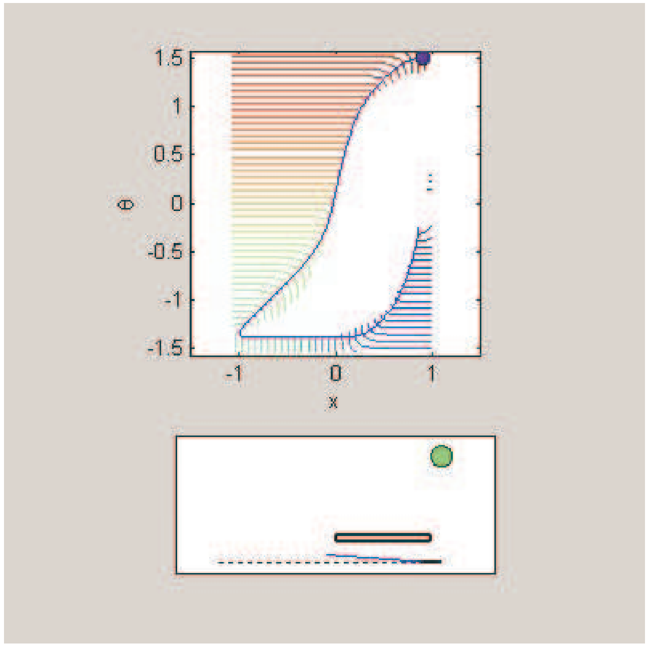


$p* = 1$

Infinity Norm

- Action bound $p = \infty$, so update formula $p^* = 1$
- Right: optimal trajectory of two joint arm under $||\cdot||_2$ (red) and $||\cdot||_\infty$ (blue)
- Below: one joint and slider arm under $||\cdot||_\infty$

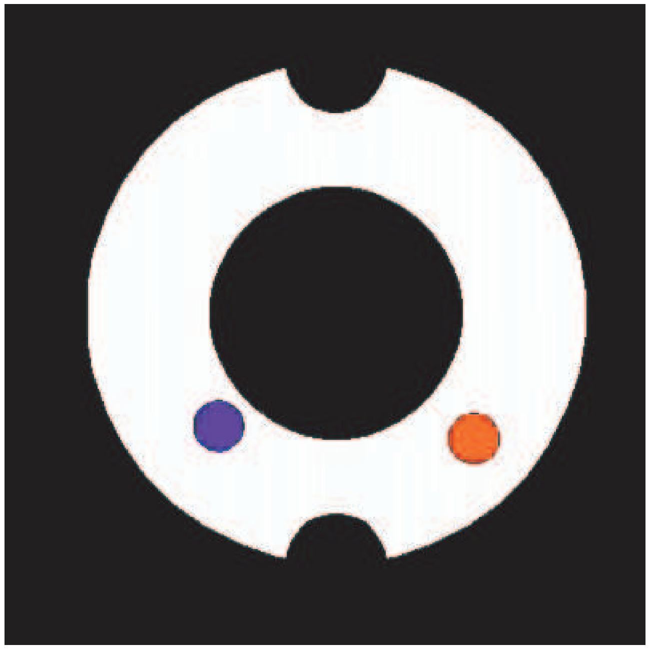


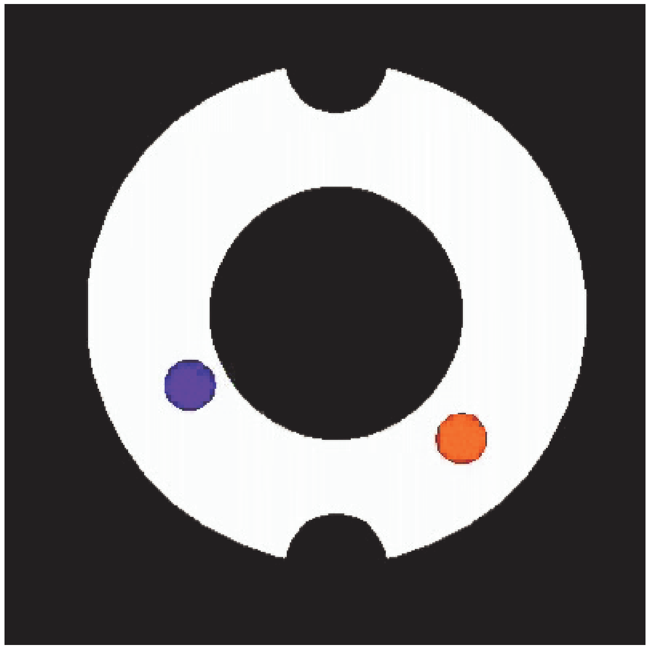


Mixtures of Norms: Multiple Vehicles

- May even be situations where action norm bounds are mixed
  - Red robot starts on right, may move any direction in 2D
  - Blue robot starts on left, constrained to 1D circular path
  - Cost encodes black obstacles and collision states
  - 2D robot action constrained in $\|\cdot\|_2$ and combined action in $\|\cdot\|_\infty$

$$\left\| \left(\left\| \left(\frac{\partial\vartheta(x)}{\partial x_1}, \frac{\partial\vartheta(x)}{\partial x_2} \right) \right\|_2, \frac{\partial\vartheta(x)}{\partial x_3} \right) \right\|_1 = c(x).$$

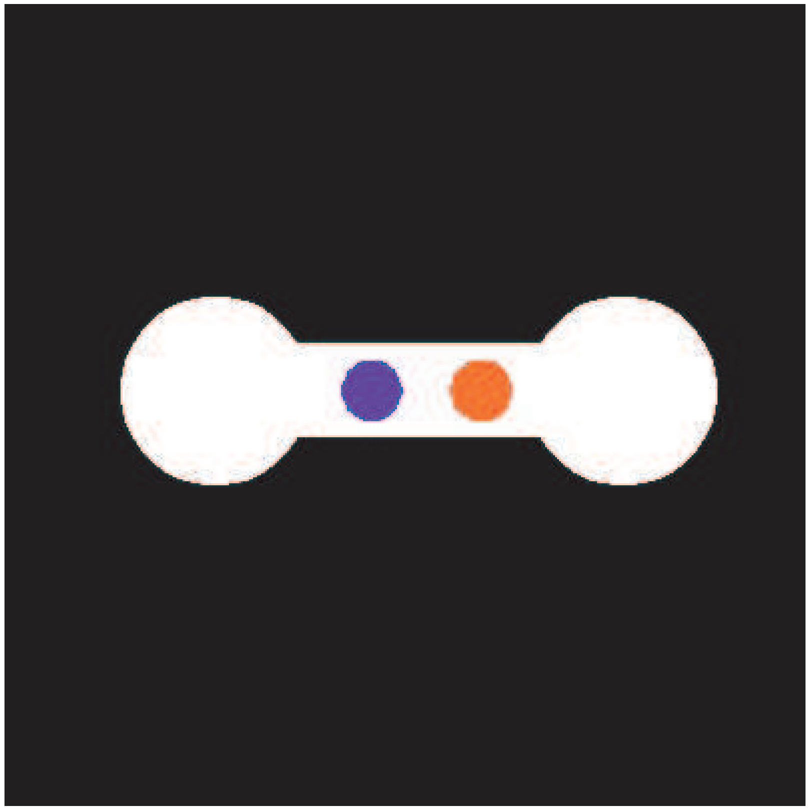




Mixtures of Norms: Multiple Vehicles

- Now consider two robots free to move in the plane

$$\left\| \left(\left\| \left(\frac{\partial \vartheta(x)}{\partial x_1}, \frac{\partial \vartheta(x)}{\partial x_2} \right) \right\|_2, \left\| \left(\frac{\partial \vartheta(x)}{\partial x_3}, \frac{\partial \vartheta(x)}{\partial x_4} \right) \right\|_2 \right) \right\|_1 = c(x).$$

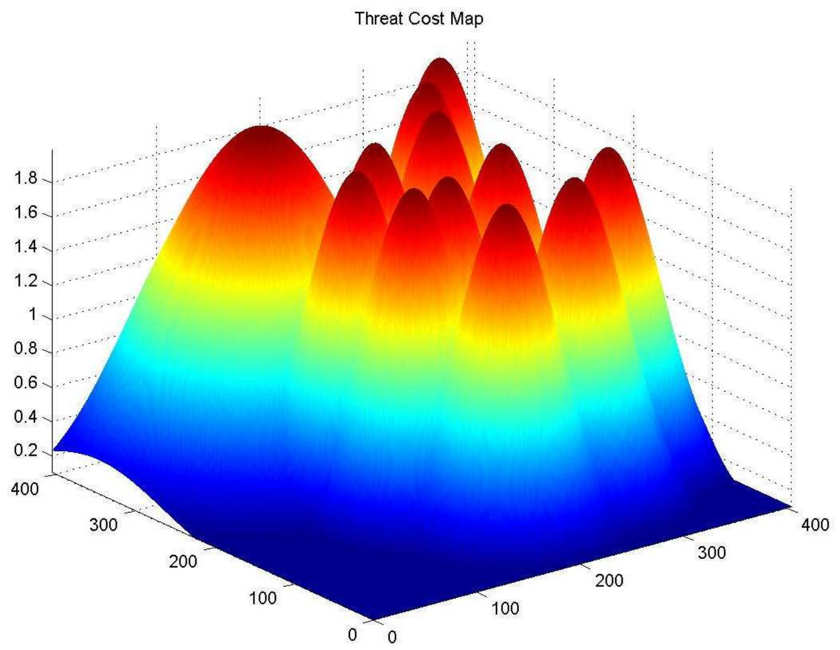


Constrained Path Planning

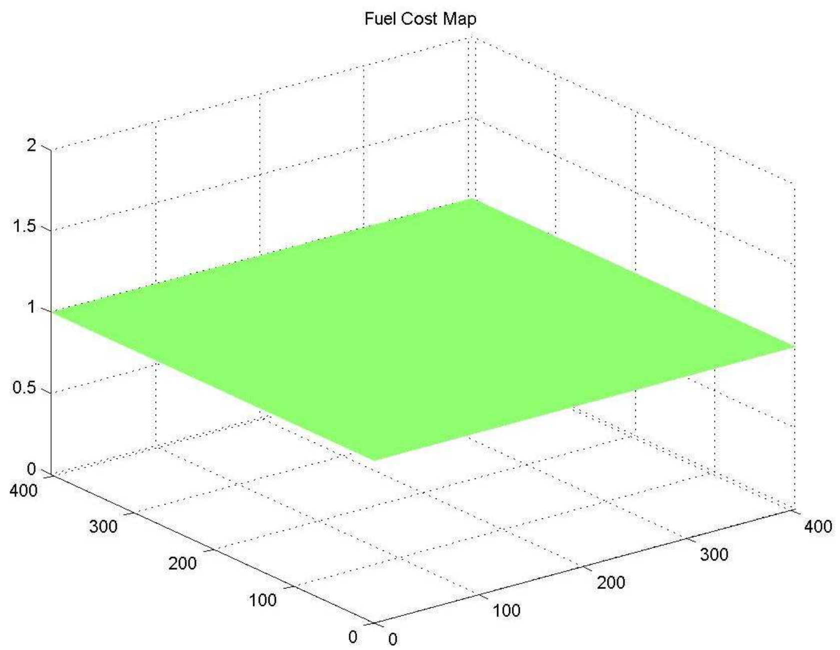
- Input includes multiple cost functions $c_i(x)$
- Possible goals:
  - Find feasible paths given bounds on each cost
  - Optimize one cost subject to bounds on the others
  - Given a feasible/optimal path, determine marginals of the constraining costs

Mitchell & Sastry, “Continuous Path Planning with Multiple Constraints,” CDC 2003

Variable cost (eg threat level)



Constant cost (eg fuel)



Path Integrals

- To determine if path $p(t)$ is feasible, we must determine

$$P_i(x) = \int_0^T c_i(p(s))ds, \text{ where } \begin{cases} p(0) = & \texttt{target}, \\ p(T) = x \end{cases}$$

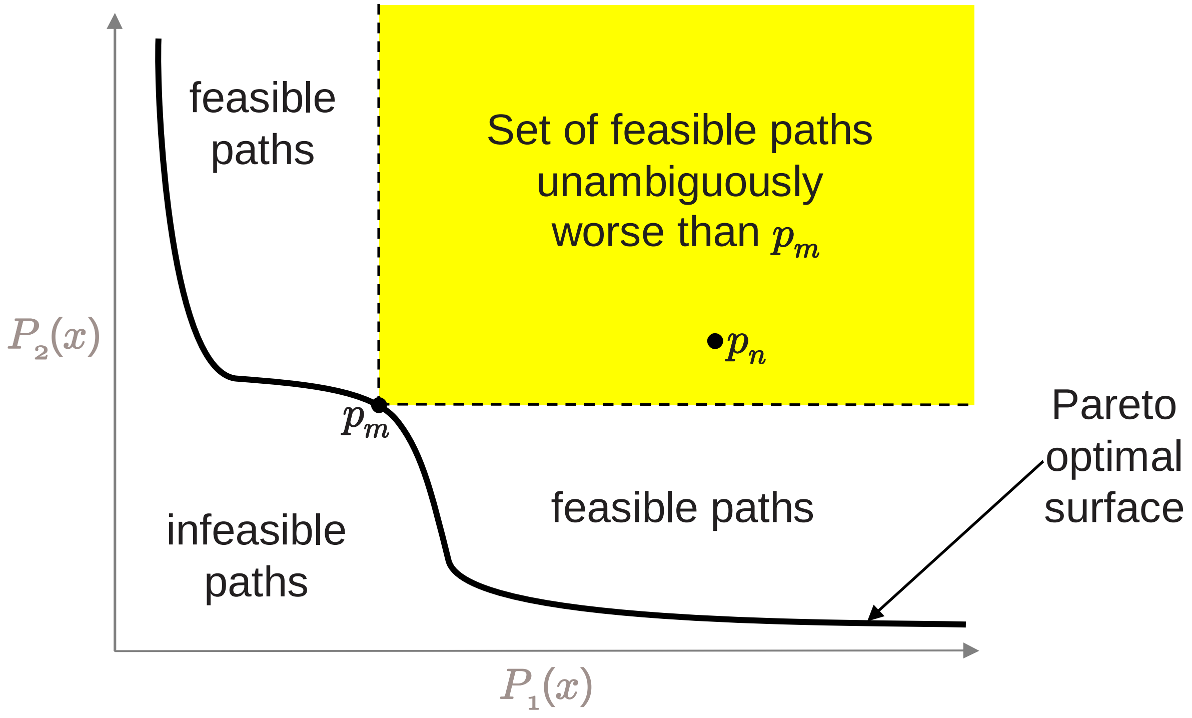
- If the path is generated from a value function $\vartheta(x)$, then path integrals can be computed by solving the PDE

$$D_x P_i(x) \cdot D_x \vartheta(x) = c_i(x)c(x)$$

- The computation of the $P_i(x)$ can be integrated into the FMM algorithm that computes $\vartheta(x)$

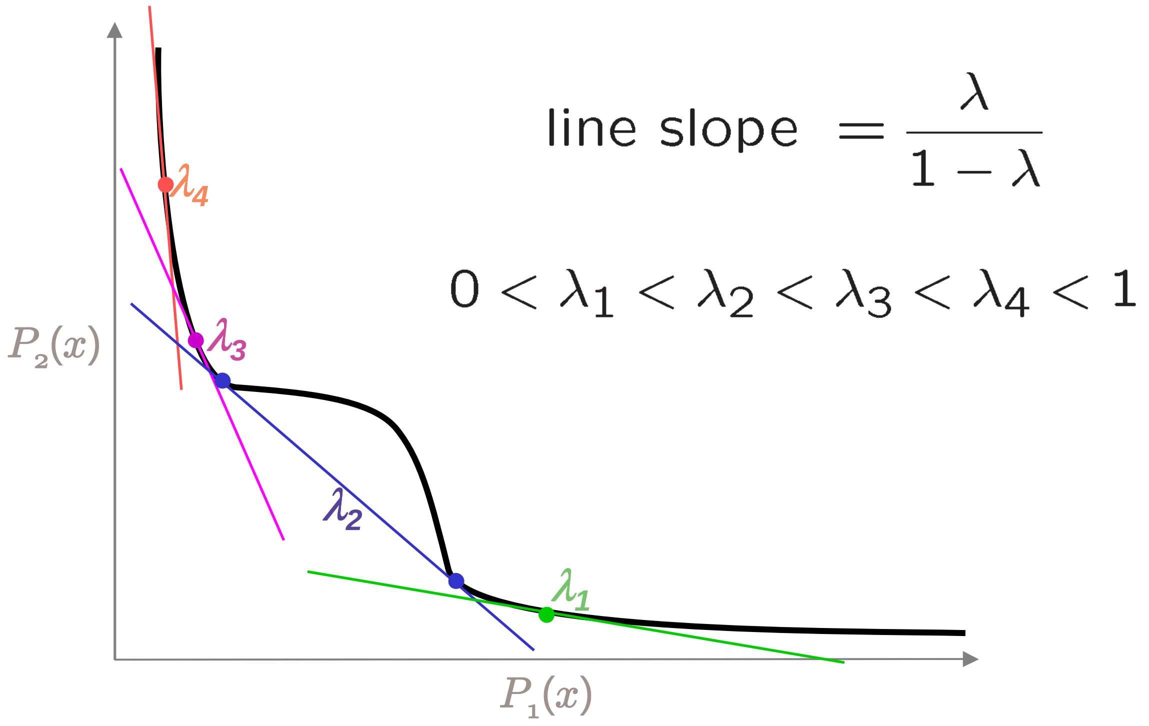
Pareto Optimality

- Consider a single point x and a set of costs $c_i(x)$
- Path p_m is unambiguously better than path p_n if
$$P_i(x; p_m) \leq P_i(x; p_n) \text{ for all } i$$
- Pareto optimal surface is the set of all paths for which there are no other paths that are unambiguously better



Exploring the Pareto Surface

- Compute value function for a convex combination of cost functions
  - For example, let $c(x) = \lambda c_1(x) + (1 - \lambda)c_2(x)$, $\lambda \in [\,0,1\,]$
- Use FMM to compute corresponding $\vartheta(x)$ and $P_i(x)$
- Constructs a convex approximation of the Pareto surface for each point x in the state space

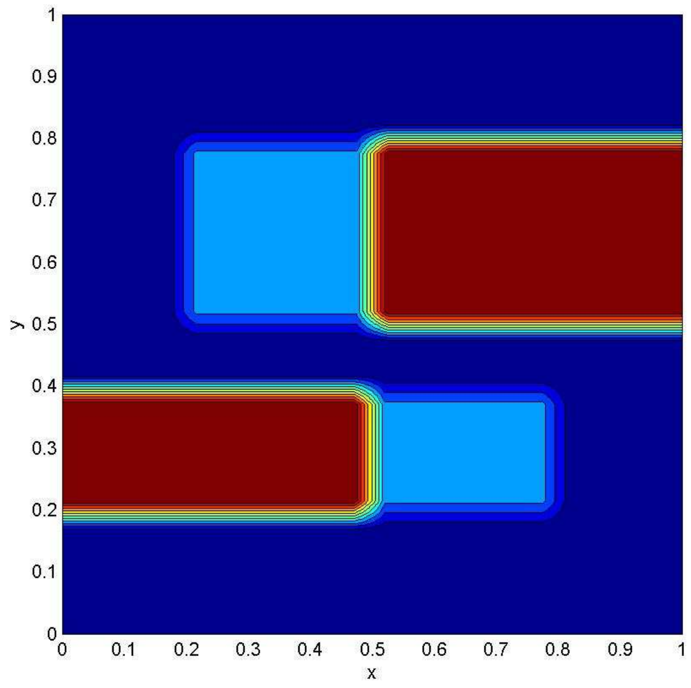


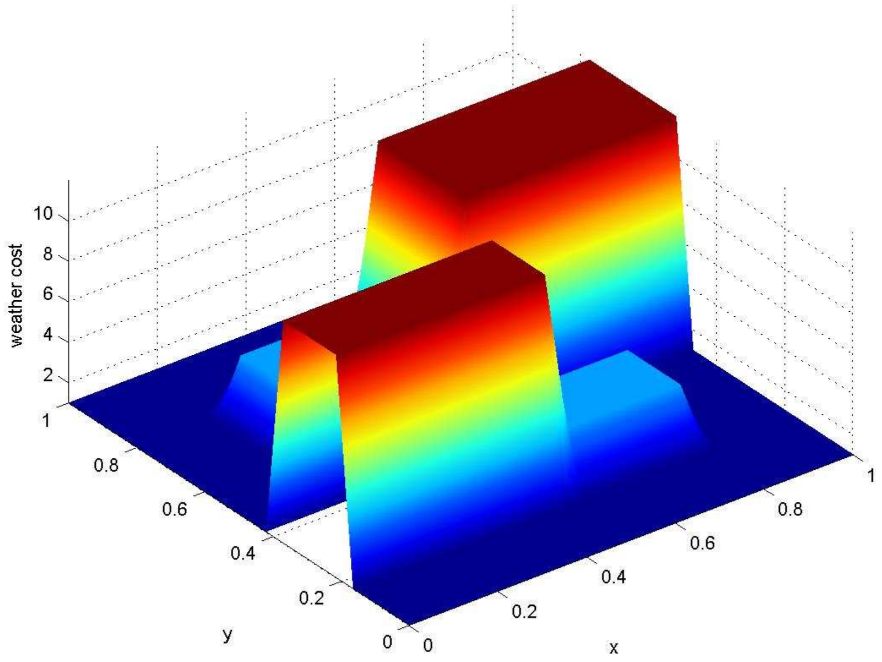
$\text{line slope} = \frac{\lambda}{1-\lambda}$
$0 < \lambda_1 < \lambda_2 < \lambda_3 < \lambda_4 < 1$
λ_4
λ_3
λ_2
λ_1
$P_2(x)$
$P_1(x)$

Constrained Path Planning Example

- Plan a path across Squaraguay
  - From Lowerleftville to Upper Right City
  - Costs are fuel (constant) and threat of a storm

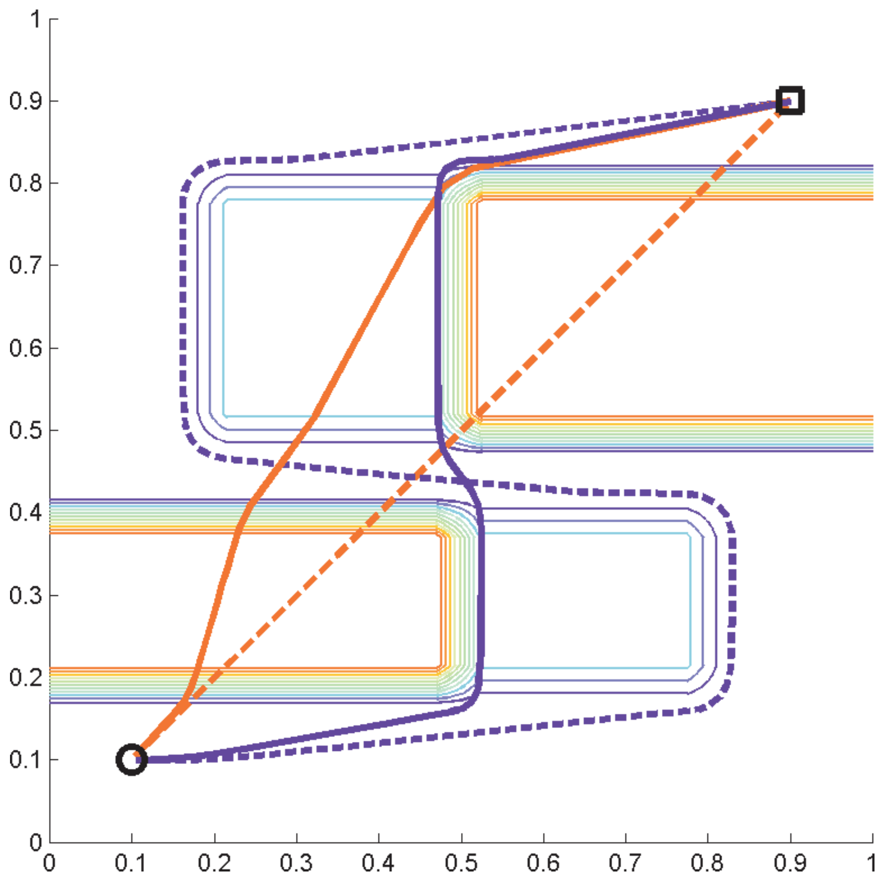
Weather cost (two views)





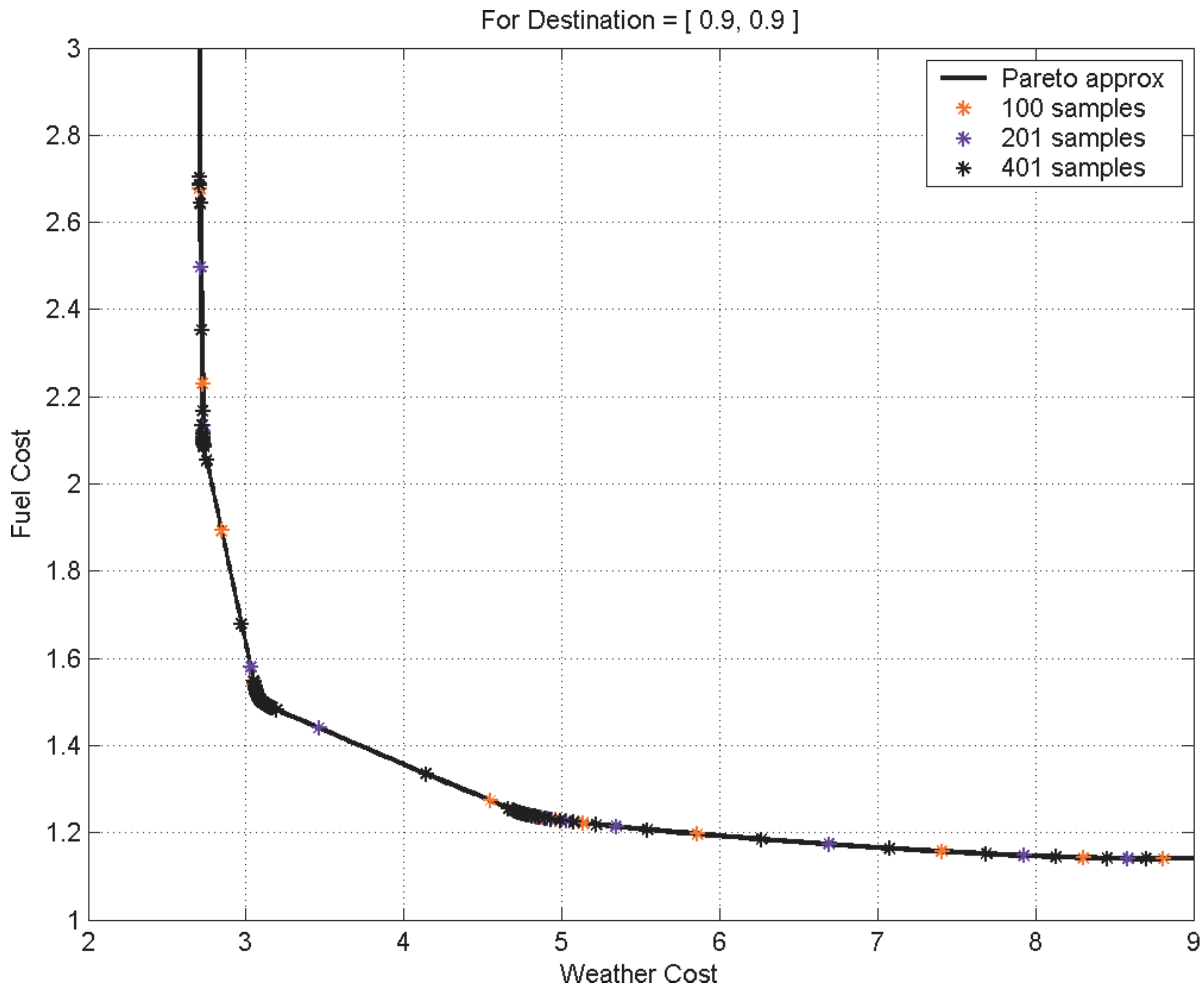
Weather and Fuel Constrained Paths

line type	minimize what?	fuel constraint	fuel cost	weather cost
- - - - -	fuel	none	1.14	8.81
———	weather	1.3	1.27	4.55
———	weather	1.6	1.58	3.03
- - - - -	weather	none	2.69	2.71



Pareto Optimal Approximation

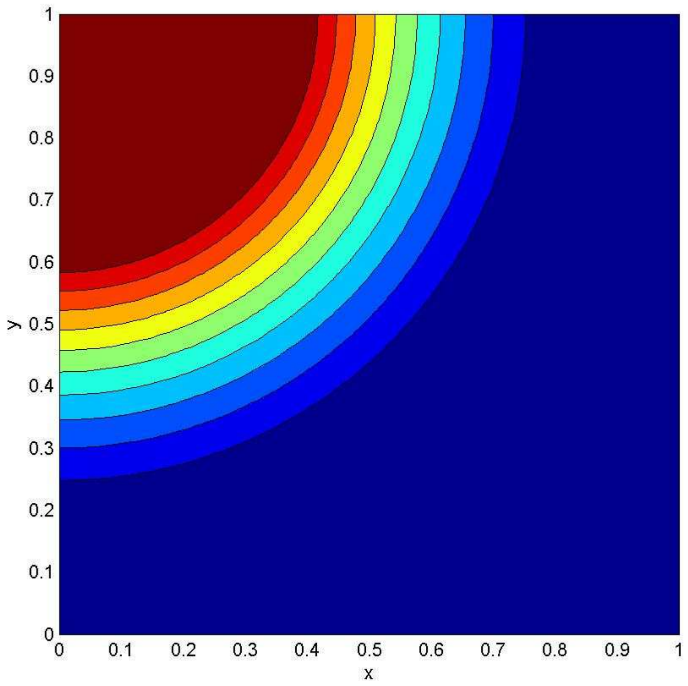
- Cost depends linearly on number of sample λ values
  - For 201^2 grid and 401 λ samples, execution time 53 seconds

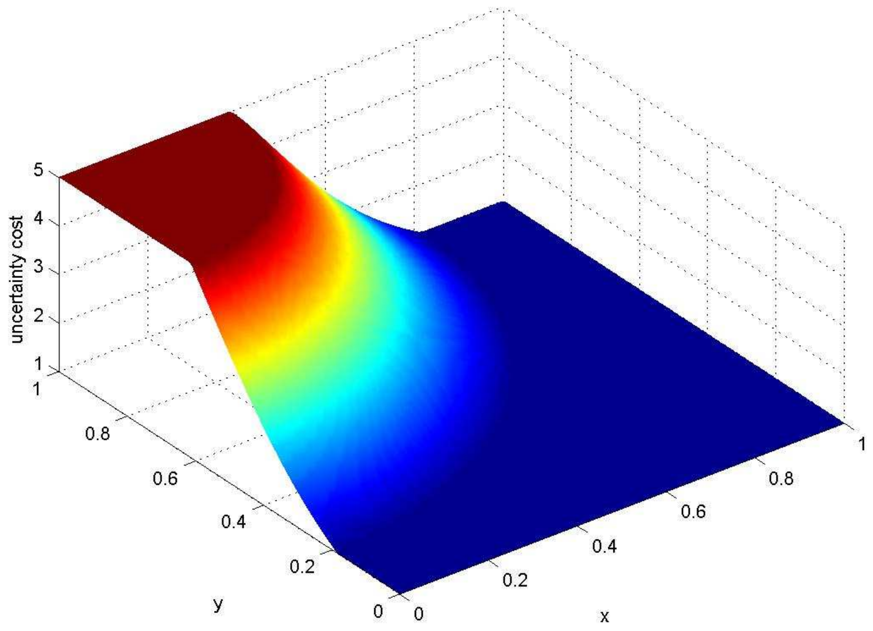


More Constraints

- Plan a path across Squaraguay
  - From Lowerleftville to Upper Right City
  - There are no weather stations in northwest Squaraguay
  - Third cost function is uncertainty in weather

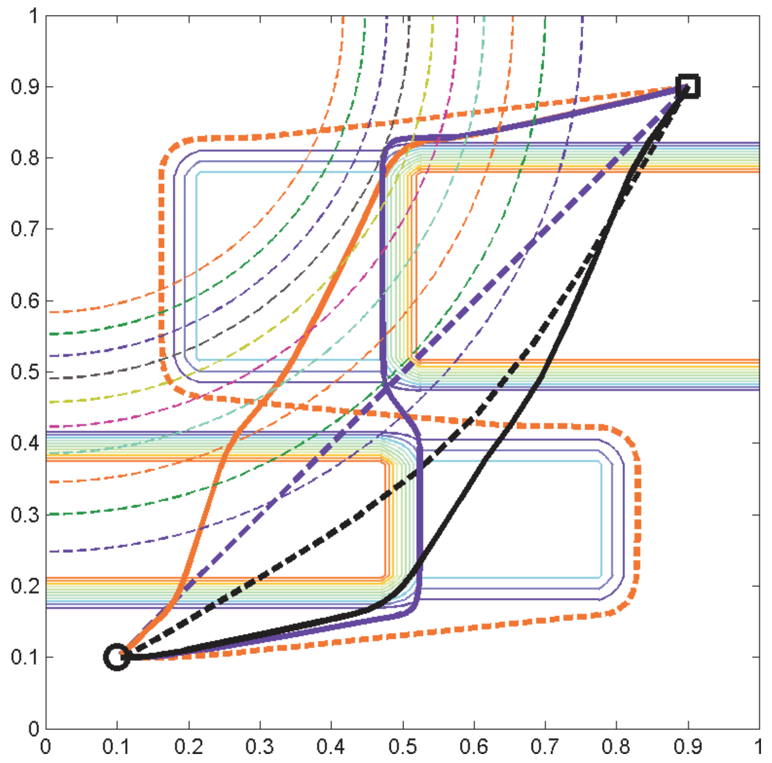
Uncertainty cost (two views)





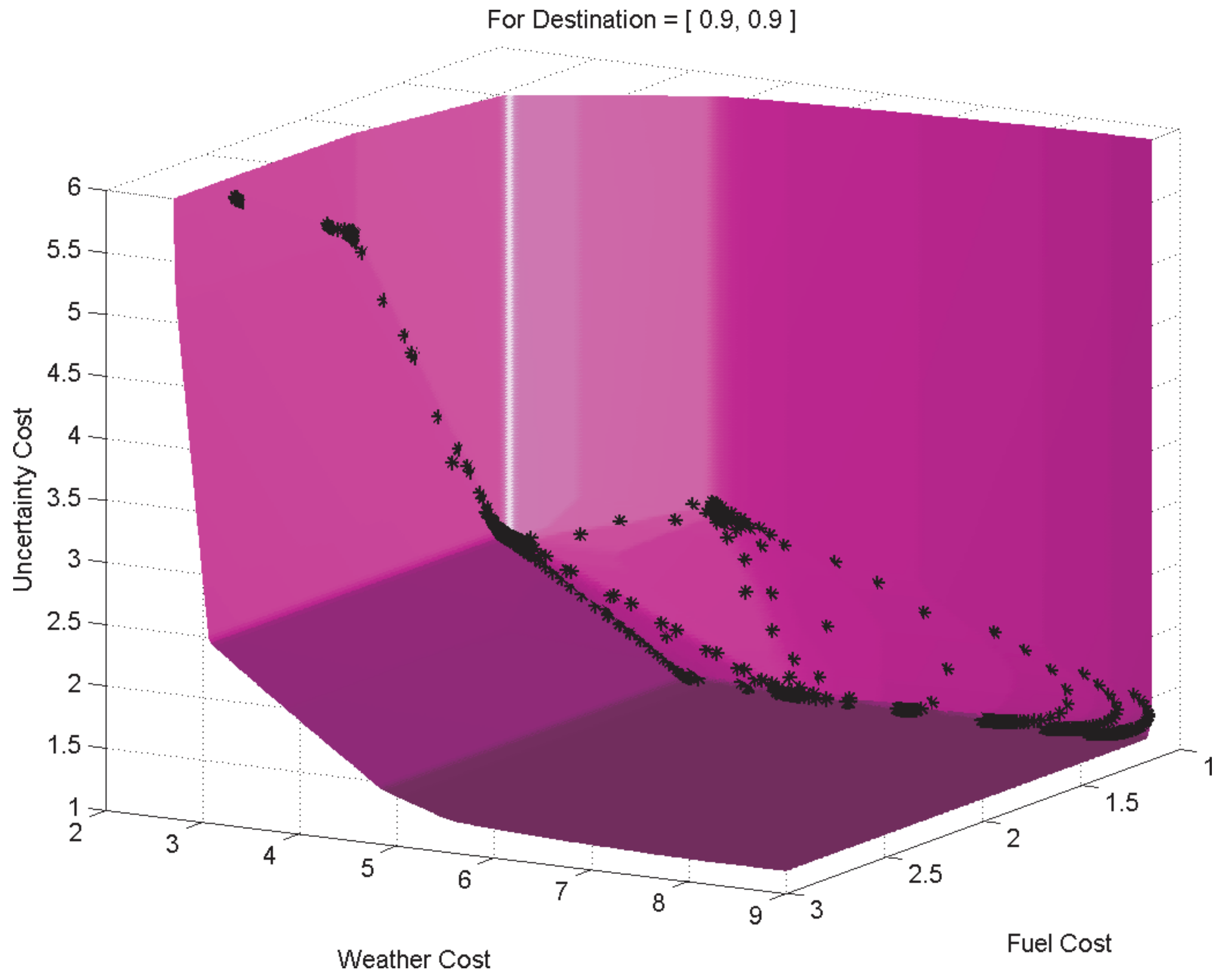
Three Costs

line type	minimize what?	fuel constraint	weather constraint	fuel cost	weather cost	uncertainty cost
- - - - -	fuel	none	none	1.14	8.81	1.50
- - - - -	weather	none	none	2.69	2.71	5.83
- - - - -	uncertainty	none	none	1.17	8.41	1.17
——	weather	1.6	none	1.60	3.02	2.84
——	weather	1.3	none	1.30	4.42	2.58
——	uncertainty	1.3	6.0	1.23	5.84	1.23



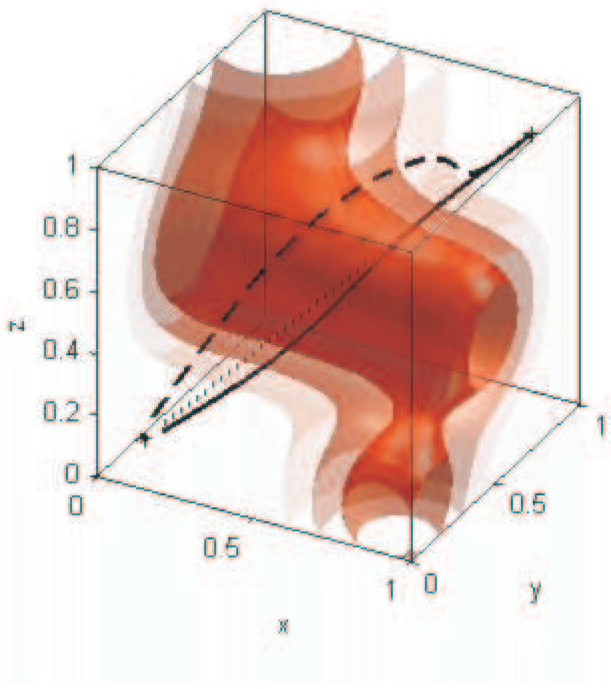
Pareto Surface Approximation

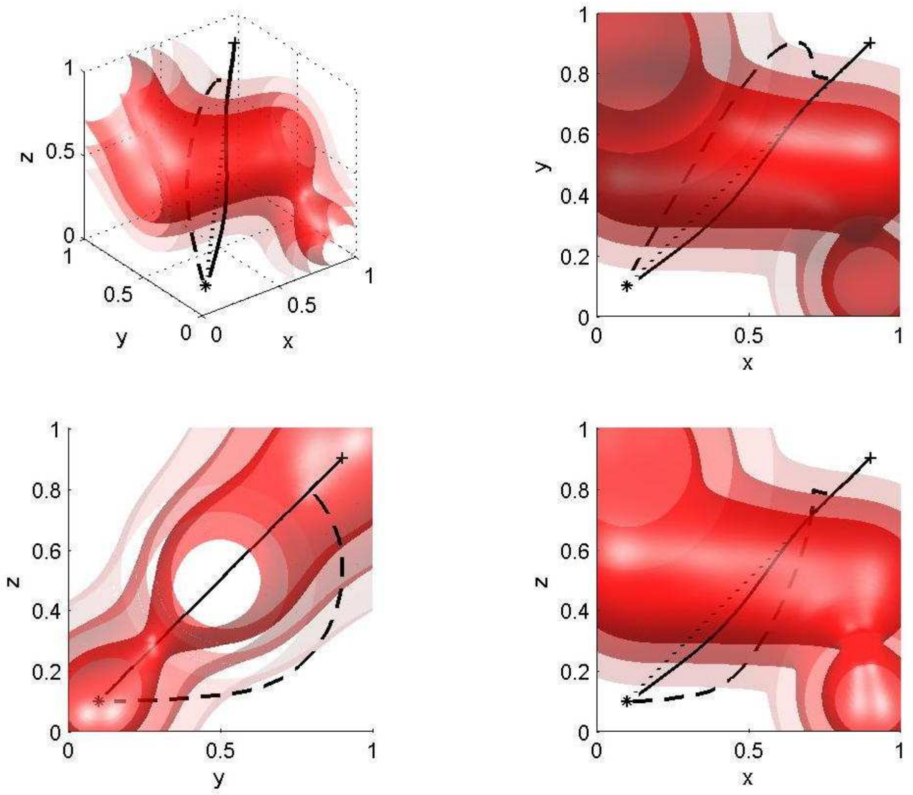
- Cost depends linearly on number of sample λ values
  - For 201^2 grid and 101^2 λ samples, execution time 13 minutes



Three Dimensions

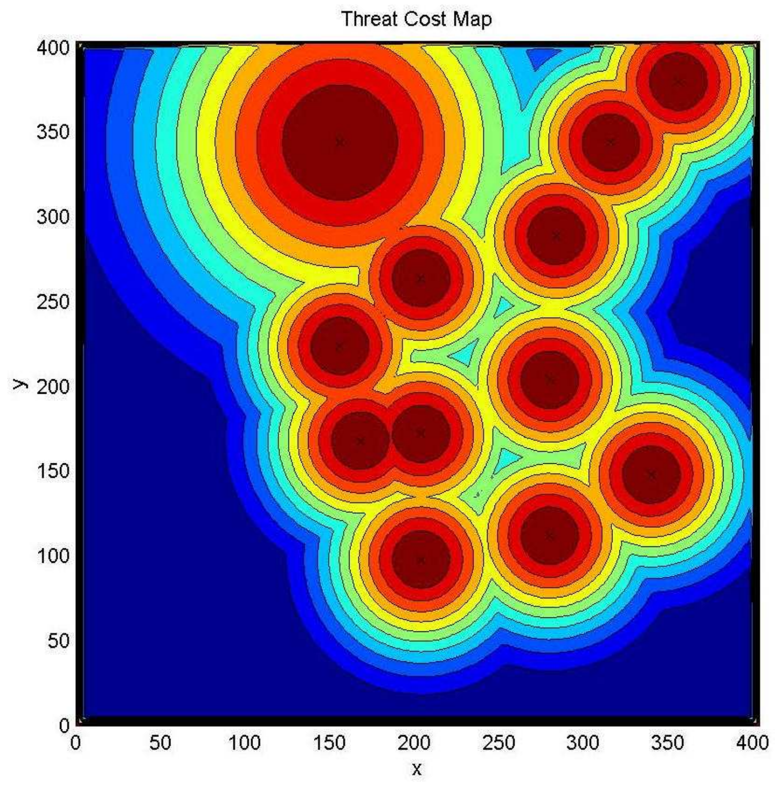
line type	minimize what?	fuel constraint	fuel cost	weather cost
- - - - -	fuel	none	1.14	3.54
— — —	weather	none	1.64	1.64
———	weather	1.55	1.55	2.00



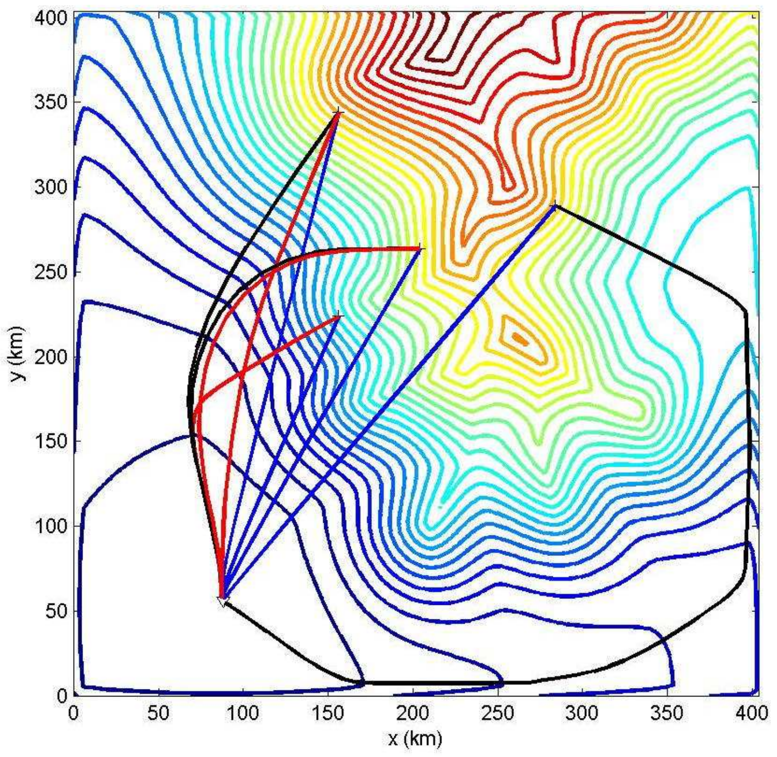


Constrained Example

- Plan path to selected sites
  - Threat cost function is maximum of individual threats
- For each target, plan 3 paths
  - minimum threat, minimum fuel, minimum threat (with fuel ≤ 300)



threat cost



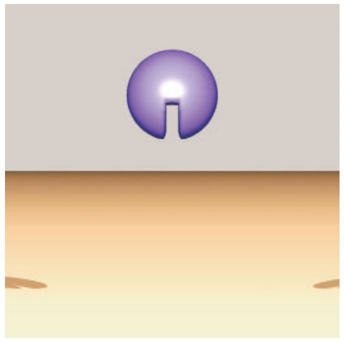
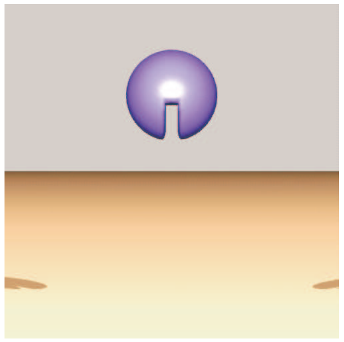
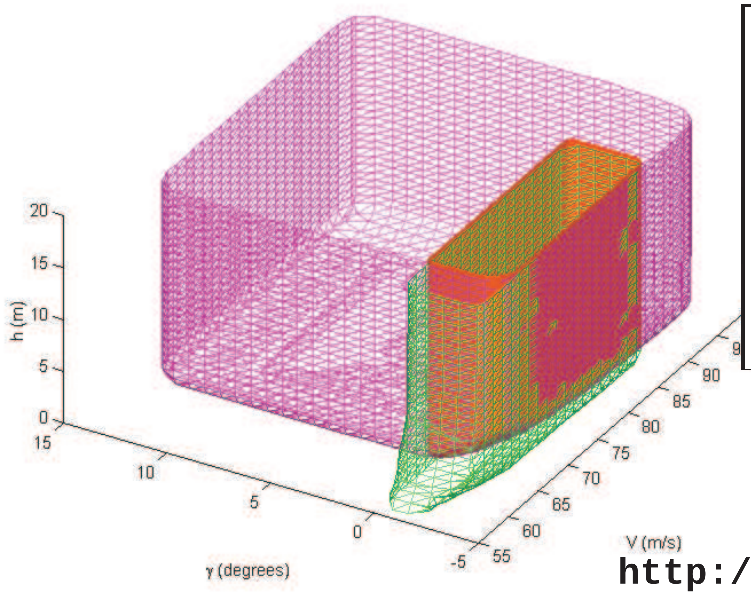
Paths (on value function)

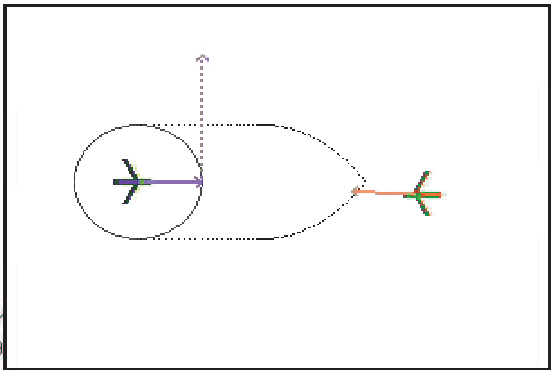
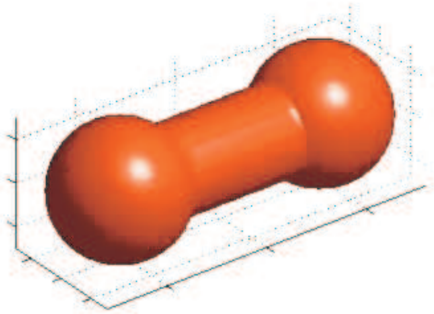
Future Work

- Fast Sweeping and Marching code
  - Python & C++
  - Interfaced to time-dependent HJ Toolbox and Matlab
- Robotic applications
  - Mesh refinement strategies
  - Integration with localization algorithms
  - Practical implementation
- Higher dimensions?
  - Taking advantage of special structure
  - Integration with suboptimal but scalable techniques

Not Discussed

- Time dependent HJ PDEs
  - Toolbox of Level Set Methods
- Reach sets
  - Safe control synthesis
  - Abstraction for verification
- Particle level sets
  - Improving volume conservation

mitchell@cs.ubc.ca
http://www.cs.ubc.ca/~mitchell

DP & HJ PDE References

- Dynamic programming
  - *Dynamic Programming & Optimal Control*, Bertsekas (3rd ed, 2007)
- HJ PDEs and viscosity solutions
  - Crandall & Lions (1983) original publication
  - Crandall, Evans & Lions (1984) current formulation
  - Evans & Souganidis (1984) for differential games
  - Crandall, Ishii & Lions (1992) "User's guide" (dense reading)
  - *Viscosity Solutions & Applications* in Springer's Lecture Notes in Mathematics (1995), featuring Bardi, Crandall, Evans, Soner & Souganidis (Capuzzo-Dolcetta & Lions eds)
  - *Optimal Control & Viscosity Solutions of Hamilton-Jacobi-Bellman Equations*, Bardi & Capuzzo-Dolcetta (1997)
  - *Partial Differential Equations*, Evans (1998)

Static HJ PDE Algorithm References

- Time-dependent transforms
  - Osher (1993)
  - Mitchell (2007): ToolboxLS documentation
- Fast Marching
  - Tsitsiklis (1994, 1995): first known description, semi-Lagrangian
  - Sethian (1996): first finite difference scheme
  - Kimmel & Sethian (1998): unstructured meshes
  - Kimmel & Sethian (2001): path planning
  - Sethian & Vladimirsky (2000): anisotropic FMM (restricted)
  - Sethian & Vladimirsky (2001, 2003): ordered upwind methods
- Fast Sweeping
  - Boue & Dupuis (1999): sweeping for MDP approximations
  - Zhao (2004), Tsai et. al (2003), Kao et. al. (2005), Qian et. al. (2007): sweeping with finite differences for static HJ PDEs

Static HJ PDE Algorithm References

- Some other related citations
  - Yatziv et. al. (2006): sloppy queue based FMM
  - Bournemann & Rasch (2006): FEM discretization
- Empirical comparisons marching vs sweeping
  - Gremaud & Kuster (2006): more numerical analysis oriented
  - Hysing & Turek (2005): more computer science oriented
- Textbooks & survey articles
  - Sethian, *SIAM Review*,1999
  - Osher & Fedkiw, *J. Computational Physics*, 2001
  - Sethian, *J. Computational Physics*, 2001
  - *Level Set Methods & Fast Marching Methods*, Sethian (2nd ed, 1999)
  - *Level Set Methods & Dynamic Implicit Surfaces*, Osher & Fedkiw (2002)

Dynamic Programming Algorithms for Planning and Robotics in Continuous Domains and the Hamilton-Jacobi Equation

For more information contact

Ian Mitchell

Department of Computer Science

The University of British Columbia

mitchell@cs.ubc.ca

http://www.cs.ubc.ca/~mitchell