Math 5587 – Lecture 21

Jeff Calder

November 16, 2016

1 The eikonal equation

1.1 Automatic maze navigation

Consider the problem of automatic maze navigation. Our maze in a two dimensional maze constructed on the plane $\mathbb{R}^2$. We have a starting point $\mathbf{x}_0 \in \mathbb{R}^2$, and a target set $\Gamma \subset \mathbb{R}^2$, which should be thought of as the 'end' of the maze. Our maze is defined by a speed function $c(\mathbf{x})$ for $\mathbf{x} \in \mathbb{R}^2$, which is the speed the robot can move at location $\mathbf{x}$. Objects the robot cannot move through, i.e., walls, are modeled by setting $c(\mathbf{x}) = 0$ in these regions. This also allows us to model the possibility that the robot may be able to move faster through some regions of the maze than others.

A path is a continuous function

$$t \mapsto \mathbf{x}(t) = (x(t), y(t)),$$

where $t \in [0, 1]$. The parameter t is artificial time, and is merely used for parameterization. It does not represent the time it takes for a robot to navigate the path $\mathbf{x}(t)$. To compute this, notice that for small $\Delta t > 0$, the distance moved along the path segment from t to $t + \delta t$ is

$$\text{dist} \approx \|\mathbf{x}'(t)\| \, \Delta t,$$

where

$$\mathbf{x}'(t) = (x'(t), y'(t)) \quad \text{and} \quad \|\mathbf{x}'(t)\| = \sqrt{x'(t)^2 + y'(t)^2}.$$

Since the robot moves at speed $c(\mathbf{x}(t))$, the time taken to traverse the segment of the path from t to $t + \Delta t$ is

$$\frac{\text{dist}}{\text{speed}} \approx c(\mathbf{x}(t))^{-1} \|\mathbf{x}'(t)\| \, \Delta t.$$

Hence, the time taken to navigate the entire path $\mathbf{x}(t)$ by the robot is

$$T(\mathbf{x}(\cdot)) := \int_0^1 c(\mathbf{x}(t))^{-1} \|\mathbf{x}'(t)\| \, dt.$$

If the path passes through a region where $c = 0$, the time T is automatically ∞. In practice, we might set c to be a very small, but nonzero number in such regions. See Figure 1 for an illustration of what the maze might look like.

The goal of maze navigation is to find the shortest path from some starting point $\mathbf{x}_0 \in \mathbb{R}^2$ to the target set Γ. Hence, the maze navigation problem can be posed as

$$\min \Big\{ T(\mathbf{x}(\cdot)) \mid \mathbf{x}(0) = \mathbf{x}_0 \text{ and } \mathbf{x}(1) \in \Gamma \Big\}. \tag{1}$$

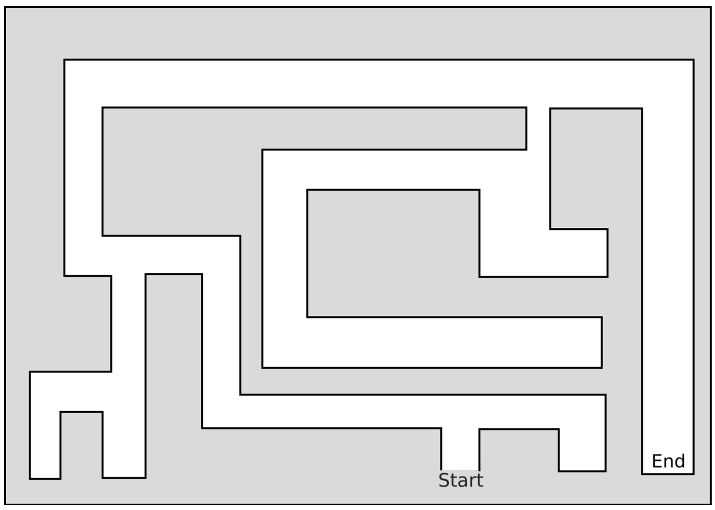

Figure 1: Example of a maze. We set the speed $c(\mathbf{x}) = 0$ in the gray region, and $c(\mathbf{x}) = 1$ in the white region constituting the maze.

1.2 The value function

Finding the optimal path, and navigating the maze, is a difficult problem to address directly. Instead, we will pass through an auxiliary problem that is connected to partial differential equations. Let us consider for each point $\mathbf{x}_0 \in \mathbb{R}^2$ the *value function* $u(\mathbf{x}_0)$ defined by

$$u(\mathbf{x}_0) := \min \Big\{ T(\mathbf{x}(\cdot)) \mid \mathbf{x}(0) = \mathbf{x}_0 \text{ and } \mathbf{x}(1) \in \Gamma \Big\}. \tag{2}$$

Thus, $u(\mathbf{x}_0)$ is the length of the shortest path from $\mathbf{x}_0$ to the target set Γ, that is, the length of the shortest path that navigates the maze starting at $\mathbf{x}_0$.

Now, you might object to this reformulation. If it was so hard to find the optimal path in the first place, then how do we have any hope of computing $u(\mathbf{x}_0)$? The point is that we now consider u as a *function* of the starting position $\mathbf{x}_0$, and we shall see shortly that u satisfies a partial differential equation. This allows us to compute u *without* any knowledge of optimal paths. Actually, the optimal paths are encoded into u, and a simple dynamic programming approach can extract the optimal path through the maze from the value function u.

Let us now derive a partial differential equation that u satisfies. First, we know that

$$u(\mathbf{x}) = 0 \quad \text{for all} \quad \mathbf{x} \in \Gamma, \tag{3}$$

since if we start in the target set, we need not move anywhere! This will be the boundary condition for our PDE. Let us also assume the speed function c is continuous, and the target set Γ is closed.

Consider a point $\mathbf{x}_0 \in \mathbb{R}^2$ that is not in the target set, and let $r > 0$ such that the ball $B(\mathbf{x}_0, r)$ of radius $r > 0$ centered at $\mathbf{x}_0$ has *no* overlap with the target set. We define

$$A(\mathbf{x}_0, \mathbf{x}_1) := \min \Big\{ T(\mathbf{x}(\cdot)) \mid \mathbf{x}(0) = \mathbf{x}_0 \text{ and } \mathbf{x}(1) = \mathbf{x}_1. \Big\}. \tag{4}$$

Then we have the following *dynamic programming principle*, whose proof is almost immediate.

$$u(\mathbf{x}_0) = \min_{\|\mathbf{x}_1 - \mathbf{x}_0\| = r} \Big\{ A(\mathbf{x}_0, \mathbf{x}_1) + u(\mathbf{x}_1) \Big\}. \tag{5}$$

The dynamic programming principle says that instead of finding an optimal path from $\mathbf{x}_0$ to the target Γ, we can split the path up into one piece that goes from $\mathbf{x}_0$ to a point $\mathbf{x}_1$ on the boundary of the ball $B(\mathbf{x}_0, r)$, and a second piece that goes from $\mathbf{x}_1$ to the target. If we minimize over the location of the point $\mathbf{x}_1$, and the two paths described above, we get the same optimal path. A rigorous proof can be constructed along these lines.

Since c is continuous, we can choose $r > 0$ smaller, if necessary, so that the approximation

$$c(\mathbf{x}_1) \approx c(\mathbf{x}_0) \quad \text{for all} \quad \mathbf{x}_1 \in B(\mathbf{x}_0, r)$$

is as good as we like. Now, when r is small, the optimal path in the definition of $A(\mathbf{x}_0, \mathbf{x}_1)$ should be a roughly a straight line of length r from $\mathbf{x}_0$ to $\mathbf{x}_1$. The time taken to traverse this straight line is then approximately $c(\mathbf{x}_0)^{-1} r$. Therefore, when $r > 0$ is small and $\|\mathbf{x}_0 - \mathbf{x}_1\| = r$

$$A(\mathbf{x}_0, \mathbf{x}_1) \approx c(\mathbf{x}_0)^{-1} r.$$

Making this approximation in the dynamic programming principle yields

$$u(\mathbf{x}_0) \approx \min_{\|\mathbf{x}_1 - \mathbf{x}_0\| = r} \Big\{ c(\mathbf{x}_0)^{-1} r + u(\mathbf{x}_1) \Big\}.$$

Let us rearrange this a bit

$$\min_{\|\mathbf{x}_1 - \mathbf{x}_0\| = r} \Big\{ c(\mathbf{x}_0)^{-1} r + u(\mathbf{x}_1) - u(\mathbf{x}_0) \Big\} \approx 0.$$

Now, recall we have the Taylor series approximation

$$u(\mathbf{x}_1) - u(\mathbf{x}_0) = \nabla u(\mathbf{x}_0) \cdot (\mathbf{x}_1 - \mathbf{x}_0) + O(r^2).$$

Inserting the Taylor series yields

$$\min_{\|\mathbf{x}_1 - \mathbf{x}_0\| = r} \Big\{ c(\mathbf{x}_0)^{-1} r + \nabla u(\mathbf{x}_0) \cdot (\mathbf{x}_1 - \mathbf{x}_0) \Big\} \approx 0.$$

We now divide both sides by r and make the change of variables

$$\mathbf{v} = \frac{\mathbf{x}_1 - \mathbf{x}_0}{r}.$$

Then the dynamic programming principle becomes

$$\min_{\|\mathbf{v}\| = 1} \Big\{ c(\mathbf{x}_0)^{-1} + \nabla u(\mathbf{x}_0) \cdot \mathbf{v} \Big\} \approx 0.$$

The optimal choice of $\mathbf{v}$ is $\mathbf{v} = -\frac{\nabla u(\mathbf{x}_0)}{\|\nabla u(\mathbf{x}_0)\|}$, which yields

$$c(\mathbf{x}_0)^{-1} - \|\nabla u(\mathbf{x}_0)\| \approx 0.$$

Sending $r \to 0$, these approximations become exact, and we find that

$$\boxed{\left.\begin{aligned} c(\mathbf{x})\|\nabla u(\mathbf{x})\| &= 1, \quad \text{if } \mathbf{x} \in \mathbb{R}^2 \setminus \Gamma \\ u(\mathbf{x}) &= 0, \quad \text{if } \mathbf{x} \in \Gamma. \end{aligned}\right\}} \tag{6}$$

This nonlinear partial differential equation is called the *eikonal* equation. The eikonal equation also shows up in wave propagation, geometric optics, computer vision, computational fluid dynamics, and many other areas. The point of this reformulation is that instead of trying to find an optimal path, we can solve the eikonal equation (6), which turns out to be much easier in practice.

Remark 1. The eikonal equation (6) does not in general have a continuously differentiable solution. For example, let $c \equiv 1$, and let $\Gamma = \{\mathbf{x} \in \mathbb{R}^2 \mid \|\mathbf{x}\| = 1\}$ be the unit sphere. Suppose a continuously differentiable solution of (6) exists inside the ball $B = \{\mathbf{x} \in \mathbb{R}^2 \mid \|x\| < 1\}$. If $u(\mathbf{x}) > 0$ for some $\mathbf{x} \in B$, then u attains its maximum at some interior point $\mathbf{x}_0 \in B$, and $\nabla u(\mathbf{x}_0) = 0$. This is at odds with the eikonal equation (6). Hence $u(\mathbf{x}) \leq 0$ for all $\mathbf{x} \in B$. Likewise, we can show that u cannot have a negative minimum in B, so $u(\mathbf{x}) \geq 0$ in B, and we get $u \equiv 0$ in B. However, such a function u clearly cannot solve the eikonal equation! Hence, we need to interpret the eikonal equation (6) in some weaker sense if we wish to obtain existence of a solution. The appropriate weak notion of solution is provided by the theory of *viscosity solutions*, which allows merely continuous functions to solve nonlinear partial differential equations [1, 2]. We will ignore these issues in the rest of the lecture.

1.3 Navigating the maze

Once we solve the eikonal equation (6) for the value function u, the optimal path to navigate the maze starting at $\mathbf{x}_0$ is the solution $\mathbf{x}(t)$ of the ordinary differential equation

$$\mathbf{x}'(t) = -\nabla u(\mathbf{x}(t)) \quad \text{with} \quad \mathbf{x}(0) = \mathbf{x}_0.$$

Indeed, we first note that the value of u is decreasing along $\mathbf{x}(t)$, since

$$\frac{d}{dt} u(\mathbf{x}(t)) = \nabla u(\mathbf{x}(t)) \cdot \mathbf{x}'(t) = -\|\nabla u(\mathbf{x}(t))\|^2 = -c(\mathbf{x}(t))^{-2} < 0.$$

Thus, provided there exists a path navigating the maze, so that $c \neq 0$ along the path, the path $\mathbf{x}(t)$ will eventually hit the target set Γ where $u = 0$. So there exists $T > 0$ such that $u(\mathbf{x}(T)) = 0$. We now compute

$$\begin{aligned} u(\mathbf{x}_0) = -(u(\mathbf{x}(T)) - u(\mathbf{x}(0))) &= -\int_0^T \frac{d}{ds} u(\mathbf{x}(s))\, ds \\ &= -\int_0^T \nabla u(\mathbf{x}(s)) \cdot \mathbf{x}'(s)\, ds \\ &= \int_0^T \|\nabla u(\mathbf{x}(s))\|^2\, ds \\ &= \int_0^T c(\mathbf{x}(s))^{-1} \|\nabla u(\mathbf{x}(s))\|\, ds = T(\mathbf{x}(\cdot)), \end{aligned}$$

where we used the fact that u satisfies the eikonal equation in the last line. Notice also that $\mathbf{x}$ is parameterized on the interval $[0, T]$, instead of $[0, 1]$ as we did earlier. However, the integral is independent of parameterization, and hence equal to $T(\mathbf{x}(\cdot))$ for any parameterization.

Since we have found a path $\mathbf{x}(t)$ such that $u(\mathbf{x}_0) = T(\mathbf{x}(\cdot))$, this path must be optimal. This approach is called *dynamic programming*. In the next section, we describe how to compute solutions of the eikonal equation numerically.

1.4 An upwind scheme for the eikonal equation

1.4.1 One dimensional example

A simple one dimensional version of the eikonal equation is

$$|u'(x)| = 1 \quad \text{for } 0 < x < 1 \quad \text{and } u(-1) = 0 = u(1). \tag{7}$$

This corresponds to speed $c \equiv 1$. The solution should be

$$u(x) = 1 - |x|.$$

When $x > 0$, the shortest path to the boundary $\Gamma = \{-1, +1\}$ goes to the right point $+1$, and the distance is $1 - x$. When $x < 0$, the shortest path to the boundary goes to the left point -1 and the distance is $1 + x = 1 - |x|$. Notice, as pointed out in Remark 1 that this solution u is not differentiable at $x = 0$. Hence it satisfies the PDE (an ODE here) at every point except $x = 0$. If we allow non-differentiable functions to be solutions of the PDE provided they satisfy the PDE at all points of differentiability, we loose uniqueness of solutions. In this case, there are infinitely many solutions constructed with a geometric folding argument that I gave in class. The solution $u(x) = 1 - |x|$ is the unique *viscosity solution*, as described in Remark 1.

The same issues show up in finite difference schemes for the eikonal equation (7). Let us discretize the interval $-1 \le x \le 1$ into a grid with resolution Δx, and let u_j be the numerical approximation of $u(j\Delta x)$. Let us suppose that $J = 1/\Delta x$ is an integer. Consider a scheme based on forward differences

$$|u_{j+1} - u_j| = \Delta x \quad \text{for } j = -J, \ldots, 0, \ldots, J-1$$

with $u_{-J} = 0$ and $u_J = 0$. A moments thought shows that there are $\binom{2J}{J}$ solutions of the scheme, so no unique solution. The picture (which I drew in class), is similar to the folding example illustrating non-uniqueness in the eikonal equation. As an exercise, the reader should verify that the same problems appear for backward and centered differences.

To fix this, we need to think about the flow of information in the problem. When $x > 0$, the shortest path goes to the right point $+1$ so information is flowing in from the right and we should use forward differences. Likewise, when $x < 0$ the shortest path goes to the left point -1, and we should use backward differences to capture this. This all sounds nice, but it relies on already knowing before hand what the solution looks like! Is there some other way to determine when to use forward and backward differences?

It turns out there is. Notice that when $x > 0$, the neighboring grid point with smallest value of $u(x)$ is to the right, whereas for $x < 0$ the smallest is to the left. Note also that the optimal path *must* pass through the neighboring grid point with smallest value. Hence information flows from small values of u to large values of u. So the idea is that we should pick the neighboring grid point that has the smallest value of $u(x)$, and use the one-sided difference corresponding to that direction. Another important property of the solution u is that it has no local minimums, so every grid point has a neighbor with smaller value. With this in mind, consider the scheme

$$u_j - \min\{u_{j-1}, u_{j+1}\} = \Delta x \quad \text{for } j = -J+1, \ldots, J-1.$$

We don't include absolute values since we require that u_j is larger than at least one of its neighbors, so it is larger than the minimum of both neighbors. We can also write the scheme as

$$u_j = \min\{u_{j-1}, u_{j+1}\} + \Delta x,$$

which is reminiscent of the dynamic programming principle (5). This scheme has a unique solution which is exactly equal to the exact solution $u_j = 1 - |j\Delta x| = u(j\Delta x)$.

1.4.2 The two dimensional eikonal equation: Fast marching

Motivated by the one dimensional example, let us consider the full two dimensional eikonal equation (6). To put it in a nicer form, let us square both sides:

$$c(\mathbf{x})^2(u_x(\mathbf{x})^2 + u_y(\mathbf{x})^2) = 1.$$

We can rearrange this as

$$u_x u_x + u_y u_y = c^{-2}.$$

Notice the left hand side is the directional derivative of u in the direction ∇u. Hence, the characteristics flow in the direction ∇u of steepest ascent, and the value function u grows most rapidly in the direction the characteristics flow. Hence, an upwind scheme looks in the opposite direction $-\nabla u$, and will therefore use grid points in the x and y directions where u has the smallest values.

Let us suppose we are on the domain $[-1, 1]^2$, and choose a grid resolution $\Delta x = \Delta y = h$ so that $J = 1/h$ is an integer. Let u_{ij} be the numerical approximation of $u(ih, jh)$ on the grid of spacing h, and let $c_{ij} = c(ih, jh)$. Then our upwind scheme for the eikonal equation should use forward differences in i when $u_{i+1,j} \leq u_{i-1,j}$, and backward differences when $u_{i+1,j} \geq u_{i-1,j}$ (when they are equal, both choices are equivalent). A similar statement is true for j. Hence our scheme is

$$\max\{u_{ij} - a_{ij}, 0\}^2 + \max\{u_{ij} - b_{ij}, 0\}^2 = c_{ij}^{-2} h^2 \tag{8}$$

where

$$a_{ij} = \min\{u_{i-1,j}, u_{i+1,j}\} \quad \text{and} \quad b_{ij} = \min\{u_{i,j-1}, u_{i,j+1}\}.$$

We take the maximum with 0 in the scheme to ensure that u_{ij} is larger than at least one of its neighbors, and the scheme only depends on neighbors with smaller values of u. The boundary conditions are $u_{ij} = 0$ for $(ih, jh) \in \Gamma$.

The scheme (8) can be solved efficiently (in $O(n \log n)$ time, where $n = h^{-2}$ is the number of grid points) using the *fast marching method* [3]. The idea is as follows. We have three sets of grid points: (1) active points, (2) visited points, and (3) unvisited points. Initially, the grid points in Γ where $u_{ij} = 0$ are placed in the visited points set, and are never visited again. The neighbors of these points are placed in the active points set. All other points are labeled unvisited. The initialization is completed by computing u_{ij} via the scheme (8) at each active point. This gives an initial guess of u_{ij}, which we will denote as T_{ij}. The algorithm then repeats the following steps:

- Find the grid point (i^*, j^*) in the active set for which $T_{i^*j^*}$ is smallest.
- Finalize this grid point by setting $u_{i^*j^*} = T_{i^*j^*}$ and moving (i^*, j^*) to the visited set.

- Add all neighbors of (i^*, j^*) to the active set, if they do not belong to the visited set.
- Update the values of T_{ij} for all active neighbors of (i^*, j^*).

The fast marching method visits the grid points in the correct order, from smallest to largest value of the solution u. Since the scheme (8) depends only on neighbors with values of u smaller than u_{ij}, the finalized values of u_{ij} in each step do not need to be updated in any future steps of the algorithm.

Each step of the algorithm finalizes the value u_{ij} at a single grid point, so the algorithm takes at most n steps to complete, where n is the number of grid points. We have to keep a sorted list of the values of T_{ij} at all active points, and we must be able to efficiently update this list upon the insertion of a new active point, or the removal of the smallest active point in Step 1. This can be done with a heap data structure, which stores the values T_{ij} in a binary tree with the smallest value on the top for quick removal. If the tree stores m values, then there are $O(\log(m))$ levels in the tree, and insertion and deletion operations have complexity $O(\log(m))$. Since $m \leq n$, we get the overall complexity of $O(n \log n)$ for the fast marching algorithm. We should mention that fast marching was inspired by Dijkstra's algorithm for finding the shortest path through a graph.

References

[1] 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*, 282(2):487–502, 1984.

[2] M. G. Crandall and P.-L. Lions. Viscosity solutions of Hamilton-Jacobi equations. *Transactions of the American Mathematical Society*, 277(1):1–42, 1983.

[3] J. A. Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences*, 93(4):1591–1595, 1996.