The Fast Marching Method

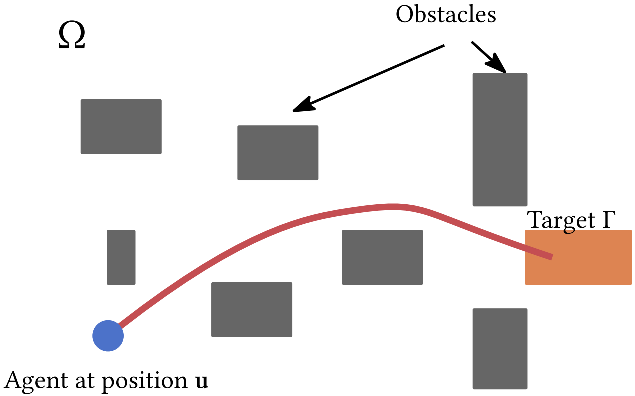
Dr. Benedikt Zönnchen

HM

11. Juni 2024

Outline

Defining the Problem



Defining the Problem

What we are looking for is a **distance function**

$$d_\Gamma : \mathbb{R}^2 \to \mathbb{R}$$

that gives us the distance to our target region Γ for any position $\mathbf{u}$ in our spatial domain Ω.

The gradient of this distance functions $-\nabla d_\Gamma$ gives us the direction in which we or the agent should move.

Defining the Problem

If there is no obstacle “in the way”, then an appropriate distance function is the Euclidean distance

$$d_\Gamma(\mathbf{u}) = \min_{\mathbf{v}\in\Gamma} \|\mathbf{u} - \mathbf{v}\|$$

and the **shortest path** from Γ to $\mathbf{u}$ follows the gradient $-\nabla d_\Gamma$:

(a) Follow $-\nabla d_\Gamma$ (b) Follow ?

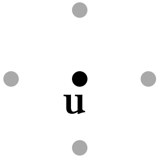
Navigating through a (Ragular) Graph

Discretization

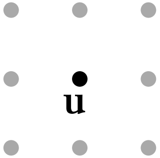
Let us first assume we discretize our domain into a regular grid

$$\Omega_h \subset \{(i \cdot h, j \cdot h) \mid i, j \in \mathbb{N}\}$$

and let us assume we have one of the following neighborhood relations



(a) Von Neumann

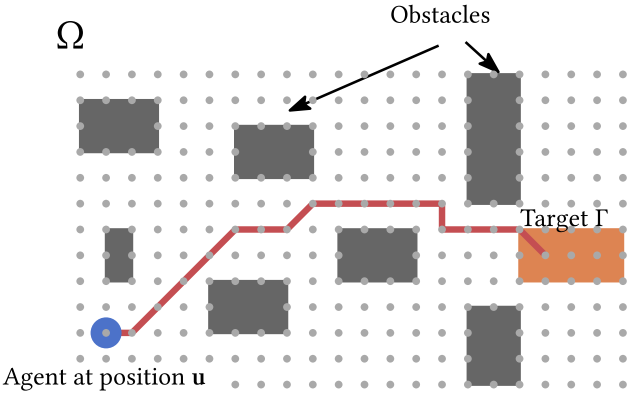


(b) Moore

such that we can only walk on the edges of the graph.

Discretization

In this case, the problems is equivalent to the problem of finding the shortest path from $\mathbf{u}$ to Γ_h on a graph!



$\Rightarrow$ Dijkstra's [3] algorithm provides a solution.

Dijkstra's Algorithm

Dijkstra's Algorithm

Strategy: Compute the shortest path for all grid points $\mathbf{u} \in \Omega$ starting at Γ_h using DIJKSTRA [3].

Observation: If $\mathbf{u}_0, \ldots, \mathbf{u}_k$ is the shortest path from $\mathbf{u}_0$ to $\mathbf{u}_k$ and $, \mathbf{u}_k, \ldots, , \mathbf{u}_m$ is the shortest path from $, \mathbf{u}_k$ to $, \mathbf{u}_m$, then

$$\mathbf{u}_0, \ldots, \mathbf{u}_m$$

is the shortest path from $\mathbf{u}_0$ to $\mathbf{u}_m$.

Dijkstra's Algorithm

Definitions:

(i) $\mathbf{u}, \mathbf{v}$: Nodes of the grid

(ii) $d_{\Gamma_h}(\mathbf{u})$: Distance between $\mathbf{u}$ and Γ_h along the shortest path

(iii) $d(\mathbf{u}, \mathbf{v})$: Distance between $\mathbf{u}$ and $\mathbf{v}$ / weight of the edge $(\mathbf{u}, \mathbf{v})$

(iv) $d_{\mathbf{u}}$: Distance between $\mathbf{u}$ and Γ_h computed by the algorithm

(v) Q: A PriorityQueue (e. g. FibonacciHeap)

Dijkstra's Algorithm

Input: Ω_h, Γ_h, d
Output: d_{Γ_h}

$d_{\mathbf{u}} \leftarrow \infty$ for all $\mathbf{u} \in \Omega_h$;
$d_{\mathbf{u}} \leftarrow 0$ for all $\mathbf{u} \in \Gamma_h$;
$\mathcal{Q} \leftarrow \{(\mathbf{u}, d_{\mathbf{u}}) \in \Gamma_h\}$;
while $\mathcal{Q} \neq \emptyset$ **do**
    $(\mathbf{u}, d_{\mathbf{u}}) \leftarrow \mathcal{Q}.\textsc{pop}()$;
    foreach *neighbour* $\mathbf{v}$ *of* $\mathbf{u}$ **do**
        $uv \leftarrow d_{\mathbf{u}} + d(\mathbf{u}, \mathbf{v})$;
        if $uv < d_{\mathbf{v}}$ **then**
            $d_{\mathbf{v}} \leftarrow uv$;
            if $(\mathbf{v}, d_{\mathbf{v}}) \in \mathcal{Q}$ **then**
                $\mathcal{Q}.\textsc{decrease}(\mathbf{v}, d_{\mathbf{v}})$;
            else
                $\mathcal{Q}.\textsc{push}(\mathbf{v}, d_{\mathbf{v}})$;
$d_{\Gamma_h} \leftarrow \{(\mathbf{u}, d_{\mathbf{u}} \mid \mathbf{u} \in \Omega_h)\}$;
return d_{Γ_h};

$\mathcal{Q}$ is sorted according to the current distance values $d_{\mathbf{v}}$:

- $\mathcal{Q}.\textsc{pop}()$, gives us the the smalles element,
- $\mathcal{Q}.\textsc{decrease}(\mathbf{v}, d_{\mathbf{v}})$ changes the element,
- and $\mathcal{Q}.\textsc{push}(\mathbf{v}, d_{\mathbf{v}})$ adds a new element

Complexity:

- Time: $\mathcal{O}(n \log(n))$
- Memory: $\mathcal{O}(n)$

Dijkstra's Algorithm

Strategy: Compute the shortest path for all grid points $\mathbf{u} \in \Omega$ to starting at Γ_h using Dijkstra [3].

Observation: If $\mathbf{u}_0, \ldots, \mathbf{u}_k, \ldots, \mathbf{u}_m$ is the shortest path from $\mathbf{u}_0$ to $\mathbf{u}_m$, then

$$\mathbf{u}_0, \ldots, \mathbf{u}_k$$

is the shortest path from $\mathbf{u}_0$ to $\mathbf{u}_k$.

Invariance: Whenever $d_{\mathbf{u}}$ gets removed from Q in line 5, it is the distance of the shortest path for all $\mathbf{u}$ to Ω_h (proof by induction).

A* Algorithm

A^* Algorithm

What if we only want to compute the shortest path from $\mathbf{u}^*$ to Γ_h.

How can we avoid computing distances for points that are located in the opposite direction?

A^* Algorithm

Strategy: Compute "towards" $\mathbf{u}^* \in \Omega_h$ first [5].

Observation (1): The Euclidean distance $\|\mathbf{u}_0 - \mathbf{u}_m\|$ is a lower bound of the distance between $\mathbf{u}_0$ and $\mathbf{u}_m$, that is,

$$\|\mathbf{u}_0 - \mathbf{u}_m\| \leq \sum_{i=0}^{m-1} d(\mathbf{u}_i, \mathbf{u}_{i+1}) = \sum_{i=0}^{m-1} \|\mathbf{u}_i - \mathbf{u}_{i+1}\|,$$

Observation (2): If

$$d_{\Gamma_h}(\mathbf{v}) + \|\mathbf{u}^* - \mathbf{v}\| > d_{\Gamma_h}(\mathbf{u}^*)$$

holds, then the shortest path from Γ_h to $\mathbf{u}^*$ does not consists of $\mathbf{v}$.

$\Rightarrow$ Sort the heap by $d_{\Gamma_h}(\mathbf{v}) + \|\mathbf{u}^* - \mathbf{v}\|$ instead of $d_{\Gamma_h}(\mathbf{v})$.

A^* Algorithm

Input: $\Omega_h, \Gamma_h, d, \mathbf{u}^*$
Output: $d_{\Gamma_h}(\mathbf{u}^*)$

$d_\mathbf{u} \leftarrow \infty$ for all $\mathbf{u} \in \Omega_h$;
$d_\mathbf{u} \leftarrow 0$ for all $\mathbf{u} \in \Gamma_h$;
/* sort by $d_\mathbf{u} + \|\mathbf{u} - \mathbf{u}^*\|$ */
$Q \leftarrow \{(\mathbf{u}, d_\mathbf{u}\}$;
while $Q \neq \emptyset$ **do**
  $(\mathbf{u}, d_\mathbf{u}) \leftarrow Q.\text{POP}()$;
  if $\mathbf{u} = \mathbf{u}^*$ **then**
    return $d_\mathbf{u}$;
  foreach *neighbour* $\mathbf{v}$ *of* $\mathbf{u}$ **do**
    $uv \leftarrow d_\mathbf{u} + d(\mathbf{u}, \mathbf{v})$;
    if $uv < d_\mathbf{v}$ **then**
      $d_\mathbf{v} \leftarrow uv$;
      if $(\mathbf{v}, d_\mathbf{v}) \in Q$ **then**
        $Q.\text{DECREASE}(\mathbf{v}, d_\mathbf{v})$;
      else
        $Q.\text{PUSH}(\mathbf{v}, d_\mathbf{v})$;

Q sorted according to the current distance values $d_\mathbf{v} + \|\mathbf{v} - \mathbf{u}^*\|$:

- $Q.\text{POP}()$, gives us the the smalles element
- $Q.\text{DECREASE}(\mathbf{v}, d_\mathbf{v})$ changes the element
- und $Q.\text{PUSH}(\mathbf{v}, d_\mathbf{v})$ adds a new element

Complexity:

- Time: $\mathcal{O}(n \log(n))$
- Memory: $\mathcal{O}(n)$

A^* Algorithms

Choice of a heuristic: The Euclidean distance $h(\mathbf{v}) := \|\mathbf{v} - \mathbf{u}^*\|$ works in our case but other heuristics might be possible. If

$$\begin{aligned} h(\mathbf{v}) &\leq d_\Gamma(\mathbf{v}) && \text{(admissible)} \\ h(\mathbf{v}) &\leq d(\mathbf{v}, \mathbf{u}) + h(\mathbf{u}) && \text{(monoton)} \end{aligned}$$

holds for each node $\mathbf{u}$ and edge $(\mathbf{u}, \mathbf{v})$, then h is **consistent** and A^* finds the shortest path without visiting nodes multipe times.

Lost advantage: If you have to compute the distance for all nodes, A^* has no advantage over DIJKSTRA.

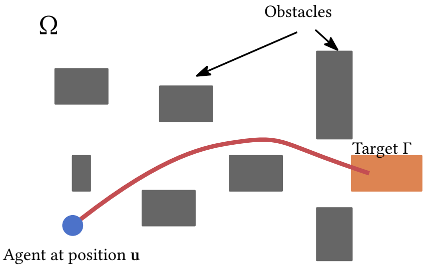
Navigating through the Continuous Space

Defining the Problem

What we are looking for a **distance function**

$$d_\Gamma : \mathbb{R}^2 \longrightarrow \mathbb{R}$$

that gives us the distance to our target region Γ for any position $\mathbf{u}$ in our spatial domain Ω.



The gradient of this distance functions $-\nabla d_\Gamma$ gives us the direction in which we or the agent should move.

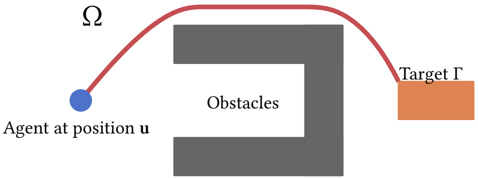
Defining the Problem

If there is no obstacle "in the way", then an appropriate distance function is the Euclidean distance

$$d_\Gamma(\mathbf{u}) = \min_{\mathbf{v}\in\Gamma} \|\mathbf{u} - \mathbf{v}\|$$

and the **shortest path** from Γ to $\mathbf{u}$ follows the gradient $-\nabla d_\Gamma$:

(a) Follow $-\nabla d_\Gamma$



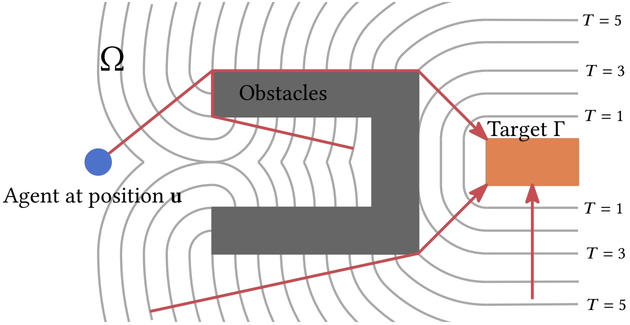
(b) Follow ?

The Eikonal Equation

The Eikonal Equation

We imagine a **wavefront** propagating with a **travel speed** $f(\mathbf{u}) = 1$. It starts at the target Γ and propagates through the region Ω.

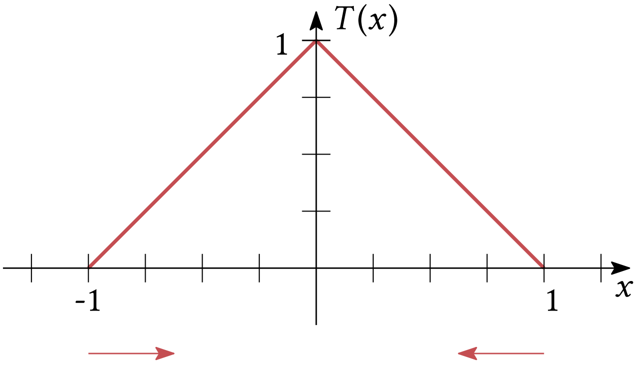
$T(\mathbf{u})$ is the **travel time** or arrival time of the **wavefront** at the location $\mathbf{u}$.



The change in travel time T (over the space) is equal to $1/f$.

The Eikonal Equation

A one-dimensional case: Let $\Omega = [-1; 1], \Gamma = \{-1, 1\}$.

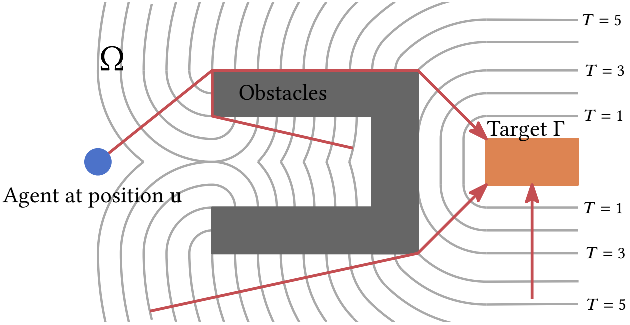


$\Rightarrow T(x) = 1 - |x|$, is the viscosity solution of the **eikonal equation**!

The Eikonal Equation

We imagine a **wavefront** propagating with a **travel speed** $f(\mathbf{u}) = 1$ from the target Γ across the region Ω.

$T(\mathbf{u})$ is the **travel time** or arrival time of the **wavefront** at the location $\mathbf{u}$.



The change in travel time T (over the space) is equal to $1/f$.

The Eikonal Equation

The **wavefront**, which propagates at **travel speed** $f(\mathbf{u}) = 1$ from the target Γ over the domain Ω, can be described by the **eikonal equation**:

$$
\begin{aligned}
\|\nabla T(\mathbf{u})\| \cdot f(\mathbf{u}) &= 1, \ \mathbf{u} \in \Omega \\
T(\mathbf{u}) &= 0, \ \mathbf{u} \in \Gamma \\
f(\mathbf{u}) &\geq 0, \ \mathbf{u} \in \Omega.
\end{aligned}
\tag{1}
$$

Remarks:

(i) It is a hyperbolic partial equation

(ii) Initial condition: $T(\mathbf{u}) = 0$ for $\mathbf{u} \in \Gamma$

(iii) For the viscosity solution T does not has to be differential everywhere

(iv) If $f = 1$ holds, then is T equal to the **geodesic distance**.

The Fast Marching Method (FMM)

The Fast Marching Method

The FastMarchingMethod [9, 10] computes a numerical solution of the eikonal equation on a discrete grid (or mesh). The algorithm imitates the propagation of the **wavefront**.

The method uses the same strategy compared to the Dijkstra but instead of computing distances between nodes it computes the **travel time** $T(\mathbf{u})$ of a wavefront that propagates over the space and not only over edges.

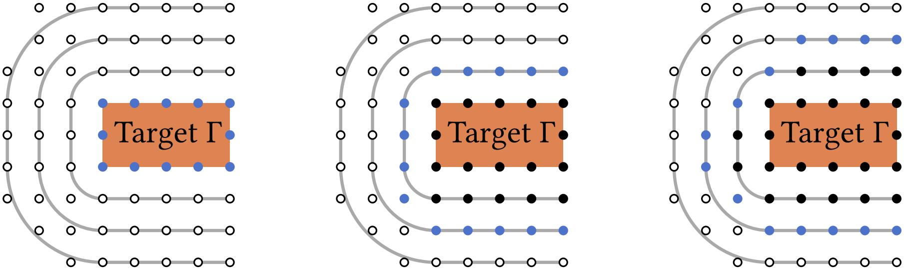
The Fast Marching Method

During the comutation, each grid point is part of exactly one of the following sets:

(i) *far*: The wavefront is far away (white)

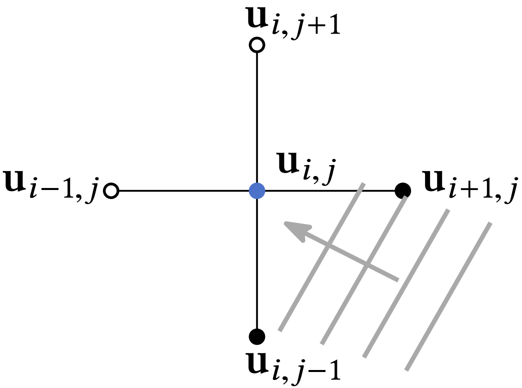
(ii) *considered*: The wavefront is close (blue)

(iii) *accepted*: The wavefront reached this point (black)



Solving the Eikonal Equation

The **wavefront** reaches every grid point $\mathbf{u}_{i,j}$ by coming from a certain direction (within one of the four quadrants):

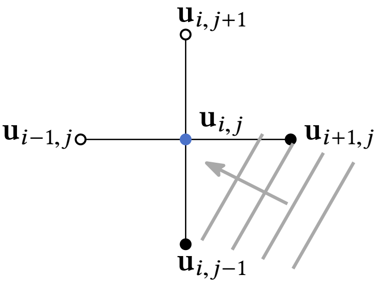


Based on the stancil of we compute the **travel time** $T(\mathbf{u}_{i,j})$.

Solving the Eikonal Equation

We can approximate ∇T by a finite difference (Taylor-expansion):

$$\frac{\partial T(\mathbf{u}_{i,j})}{\partial x} \approx D^{\pm x}_{i,j}\mathbf{u} = \frac{T(\mathbf{u}_{i\pm 1,j}) - T(\mathbf{u}_{i,j})}{\pm\Delta x}$$
$$\frac{\partial T(\mathbf{u}_{i,j})}{\partial y} \approx D^{\pm y}_{i,j}\mathbf{u} = \frac{T(\mathbf{u}_{i,j\pm 1}) - T(\mathbf{u}_{i,j})}{\pm\Delta y}.$$



If we knew that the wave arrives from right below, we only would require

$$D^{+x}_{i,j}\mathbf{u} = \frac{T(\mathbf{u}_{i+1,j}) - T(\mathbf{u}_{i,j})}{+\Delta x} \text{ und } D^{-y}_{i,j}\mathbf{u} = \frac{T(\mathbf{u}_{i,j-1}) - T(\mathbf{u}_{i,j})}{-\Delta y}.$$

Solving the Eikonal Equation

How do we arrive at an expression for $T(\mathbf{u}_{i,j})$?

$$\|\nabla T(\mathbf{u})\| \cdot f(\mathbf{u}) = 1 \tag{2}$$

can be transformed to

$$\|\nabla T(\mathbf{u})\|^2 = \frac{1}{f(\mathbf{u})^2}, \tag{3}$$

which can be further transformed into

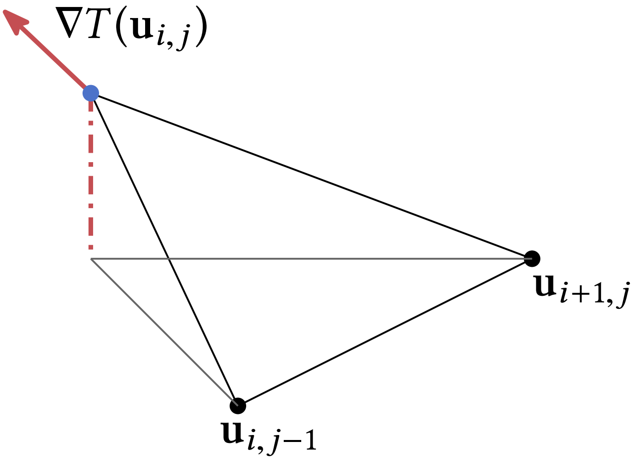
$$(D_{i,j}^{+x}\mathbf{u})^2 + (D_{i,j}^{-y}\mathbf{u})^2 = f(\mathbf{u}_{i,j})^{-2}. \tag{4}$$

(if we knew the wave arrives from right below). Therefore, we solve for $T(\mathbf{u}_{i,j})$ in the quadratic equation

$$\left(\frac{T(\mathbf{u}_{i+1,j}) - T(\mathbf{u}_{i,j})}{+\Delta x}\right)^2 + \left(\frac{T(\mathbf{u}_{i,j-1}) - T(\mathbf{u}_{i,j})}{-\Delta y}\right)^2 = f(\mathbf{u}_{i,j})^{-2}. \tag{5}$$

Solving the Eikonal Equation

The major difference to the the DIJKSTRA is the computation of the **travel time** $T(\mathbf{u})$.



Solving the Eikonal Equation

In general, we do not konw the direction from wich the wavefront arrives at $\mathbf{u}_{i,j}$. In $x-$direction it arrives from either **left** or **right** AND in y-direction from either **above** or **below**.

Right, below:

$$\left(\frac{T(\mathbf{u}_{i+1,j}) - T(\mathbf{u}_{i,j})}{+\Delta x}\right)^2 + \left(\frac{T(\mathbf{u}_{i,j-1}) - T(\mathbf{u}_{i,j})}{-\Delta y}\right)^2 = f(\mathbf{u}_{i,j})^{-2}$$

Left, above:

$$\left(\frac{T(\mathbf{u}_{i-1,j}) - T(\mathbf{u}_{i,j})}{-\Delta x}\right)^2 + \left(\frac{T(\mathbf{u}_{i,j+1}) - T(\mathbf{u}_{i,j})}{+\Delta y}\right)^2 = f(\mathbf{u}_{i,j})^{-2}$$

We assume that the wavefront arrives from the direction it arrives first.

Solving the Eikonal Equation

We assume that the **wavefront** arrives from the direction from which it arrives first at $\mathbf{u}_{i,j}$.

Therefore,

$$(D_{i,j}^{+x}\mathbf{u})^2 + (D_{i,j}^{-y}\mathbf{u})^2 = f(\mathbf{u}_{i,j})^{-2}$$

can be transformed into

$$\max\left\{D_{i,j}^{-x}\mathbf{u}, -D_{i,j}^{+x}\mathbf{u}\right\}^2 + \max\left\{D_{i,j}^{-y}\mathbf{u}, -D_{i,j}^{+y}\mathbf{u}\right\}^2 = f(\mathbf{u}_{i,j})^{-2}. \tag{6}$$

We solve this eqaution locally using Godunov's scheme [9, 11].

Solving the Eikonal Equation

We could use a more accurate approximation of ∇T useing an additional Taylor-terms. Remember:

$$f(x+h) \approx f(x) + hf'(x) + h^2 \frac{f''(x)}{2} \Rightarrow f'(x) \approx \frac{f(x+h) - f(x)}{h} - \frac{h}{2} f''(x) \tag{7}$$

For our approximation of the differentiation

$$D_{i,j}^{\pm x}\mathbf{u} = \frac{T(\mathbf{u}_{i\pm1,j}) - T(\mathbf{u}_{i,j})}{\pm\Delta x} \tag{8}$$

we get

$$(D_{i,j}^{\pm x})'\mathbf{u} \approx \frac{T(\mathbf{u}_{i\pm2,j}) - 2T(\mathbf{u}_{i\pm2,j}) + T(\mathbf{u}_{i,j})}{\pm(\Delta x)^2}. \tag{9}$$

Therefore

$$\frac{\partial T(\mathbf{u}_{i,j})}{\partial x} \approx D_{i,j}^{\pm 2x}\mathbf{u} = \frac{T(\mathbf{u}_{i\pm1,j}) - T(\mathbf{u}_{i,j})}{\pm\Delta x} - \frac{\Delta x}{2} \frac{T(\mathbf{u}_{i\pm2,j}) - 2T(\mathbf{u}_{i\pm1,j}) + T(\mathbf{u}_{i,j})}{\pm(\Delta x)^2}$$

Solving the Eikonal Equation

Therefore

$$\frac{\partial T(\mathbf{u}_{i,j})}{\partial x} \approx D^{\pm 2x}_{i,j}\mathbf{u} = \frac{T(\mathbf{u}_{i\pm 1,j}) - T(\mathbf{u}_{i,j})}{\pm\Delta x} - \frac{\Delta x}{2}\frac{T(\mathbf{u}_{i\pm 2,j}) - 2T(\mathbf{u}_{i\pm 1,j}) + T(\mathbf{u}_{i,j})}{\pm(\Delta x)^2}$$
$$= \frac{2T(\mathbf{u}_{i\pm 1,j}) - 2T(\mathbf{u}_{i,j})}{\pm 2\Delta x} - \frac{T(\mathbf{u}_{i\pm 2,j}) - 2T(\mathbf{u}_{i\pm 1,j}) + T(\mathbf{u}_{i,j})}{\pm 2\Delta x}$$
$$= \frac{-T(\mathbf{u}_{i\pm 2,j}) + 4T(\mathbf{u}_{i\pm 1,j}) - 3T(\mathbf{u}_{i,j})}{\pm 2\Delta x}.$$

The same can be computed for the y-direction:

$$\frac{\partial T(\mathbf{u}_{i,j})}{\partial y} \approx D^{\pm 2y}_{i,j}\mathbf{u} = \frac{-T(\mathbf{u}_{i,j\pm 2}) + 4T(\mathbf{u}_{i,j\pm 1}) - 3T(\mathbf{u}_{i,j})}{\pm 2\Delta y}.$$

Solving the Eikonal Equation

We still solve a quadratic equation:

$$\max\left\{D_{i,j}^{-2x}\mathbf{u}, -D_{i,j}^{+2x}\mathbf{u}\right\}^2 + \max\left\{D_{i,j}^{-2y}\mathbf{u}, -D_{i,j}^{+2y}\mathbf{u}\right\}^2 = f(\mathbf{u}_{i,j})^{-2}. \tag{10}$$

Advantage: A better rate of convergence for with respect to the grid resolution ($\Delta x, \Delta y \to 0$), since

$$f(x+h) = f(x) + hf'(x) + \mathcal{O}(h^2) = f(x) + hf'(x) + h^2\frac{f''(x)}{2} + \mathcal{O}(h^3)$$

Disadvantage: Possible unintended smoothing of singularities.

The Fast Marching Method

```
T_u ← ∞ for all u ∈ Ω;
T_u ← 0 for all u ∈ Γ;
B ← ∅ // reached points
Q ← {(u, T_u) | u ∈ Γ} // considered points
while Q ≠ ∅ do
    (u, T_u) ← Q.POP();
    foreach neighbor v of u with v ∉ B do
        T_v ← SOLVEEIKONAL(v);
        if (v, T_v) ∈ Q then
            Q.DECREASE(v, T_v);
        else
            Q.PUSH(v, T_v);
    B ← B ∪ {u};
T ← {(u, T_u)};
return T;
```

The Fast Marching Method

$\mathcal{Q}$ is a PriorityQueue sorted according to $T_{\mathbf{u}}$

- $\mathcal{Q}$.pop(), returns the point with the smalles arrival time,
- $\mathcal{Q}$.decrease$(\mathbf{u}, T_{\mathbf{u}})$ changes the element,
- and $\mathcal{Q}$.push$(\mathbf{u}, T_{\mathbf{u}})$ adds an element
- SolveEikonal$(\mathbf{u})$ solves lokal solution of Eq. (10) or Eq. (6).

Complexity: (for n nodes)

- Time: $\mathcal{O}(n \log(n))$
- Memory: $\mathcal{O}(n)$

The Fast Marching Method

Hints for your implementation:

- You find a good description in [1].
- To quickly check whether a grid point is *far*, *considered*, or *reached*, use a **flag** (not a set).
- You can also initialize points around Γ with, for example, the value of the Euclidean distance and insert them into Q.
- You can gain some performance by skipping DECREASE and using only PUSH (i.e., allowing duplicate entries $\Rightarrow$ you need to adjust the algorithm slightly, see [8])

The Fast Marching Method

Properties:

+ Very fast sequential algorithm for all types of waves
+ Especially fast for 'narrow' waves
- Difficult to parallelize, attempts [6, 12]
- Requires complicated/unstructured PriorityQueue
- Does not exploit the parallelism of wavefront propagation

Alternative methods:

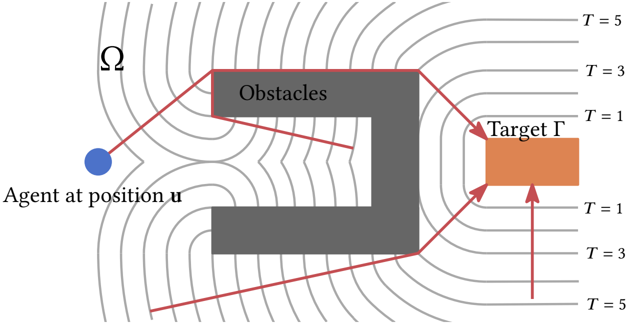
- FastSweepingMethod [13], suitable only for very 'simple' waves
- FastIterativeMethod [7], particularly suitable for 'broad' waves
- InformedFastIterativeMethod (my dissertation), suitable for repeated calculations of slightly changing waves.

In [2, 4] you can find comparisons of different methods.

Modelling using the Traveling Speed Function

Modelling using the Traveling Speed Function

How do we ensure that agents do not walk directly along the walls?



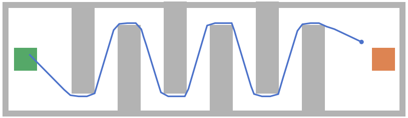
Tip: Reduce the travel speed of the wave f near obstacles!

Modelling using the Traveling Speed Function

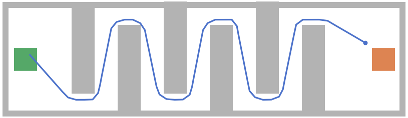
For example, let $d_W(\mathbf{u})$ be the Euclidean distance to the nearest obstacle/wall, then

$$f(\mathbf{u}) = \begin{cases} 1/(2 - (d_W(\mathbf{u})/\delta_W)) & \text{if } d_W(\mathbf{u}) < \delta_W \\ 1 & \text{otherwise.} \end{cases}$$

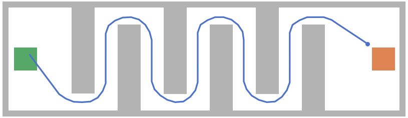
might be suitable.



(a) $\delta_W = 0.2$ meters

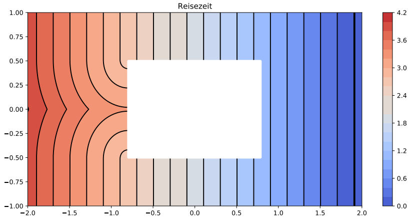


(b) $\delta_W = 0.5$ meters

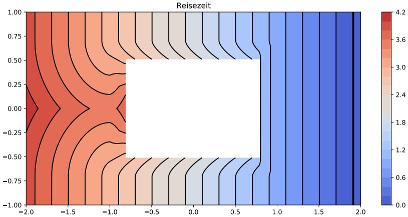


(c) $\delta_W = 1.0$ meters

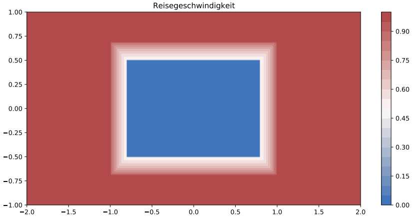
Modelling using the Traveling Speed Function



(a) $T(\mathbf{u})$ for $f(\mathbf{u}) = 1$



(b) $T(\mathbf{u})$ for $f(\mathbf{u}) \leq 1$



(c) $f(\mathbf{u}) \leq 1$

References I

[1] J. Andreas Bærentzen. On the implementation of fast marching methods for 3d lattices. Technical report, Technical University of Denmark, 2001.

[2] A. Capozzoli, C. Curcio, A. Liseno, and S. Savarese. A comparison of fast marching, fast sweeping and fast iterative methods for the solution of the eikonal equation. In *2013 21st Telecommunications Forum Telfor (TELFOR)*, pages 685–688, 2013.

[3] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959. doi: 10.1007/BF01386390.

[4] Javier V. Gómez, David Álvarez, Santiago Garrido, and Luis Moreno. Fast methods for eikonal equations: an experimental survey. *CoRR*, abs/1506.03771, 2015.

[5] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. doi: 10.1109/TSSC.1968.300136.

[6] M. Herrmann. A domain decomposition parallelization of the fast marching method. Technical report, Stanford Univ. Center for Turbulence Research; Stanford, CA, United States, 2003.

[7] Won-Ki Jeong and Ross T. Whitaker. A fast iterative method for eikonal equations. *SIAM Journal of Scientific Computing*, 30(5):2512–2534, 2008. doi: 10.1137/060670298.

[8] M. W. Jones, J. A. Baerentzen, and M. Sramek. 3d distance fields: a survey of techniques and applications. *IEEE Transactions on Visualization and Computer Graphics*, 12(4):581–599, July 2006. ISSN 1077-2626. doi: 10.1109/TVCG.2006.56.

References II

[9] R. Kimmel and J. A. Sethian. Computing geodesic paths on manifolds. *Proceedings of the National Academy of Sciences of the United States of America*, 95(15):8431–8435, 1998. doi: 10.1073/pnas.95.15.8431.

[10] J. A. Sethian. *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science.* Cambridge University Press, Cambridge, 1999.

[11] J. A. Sethian and A. Vladimirsky. Fast methods for the eikonal and related Hamilton-Jacobi equations on unstructured meshes. *Proceedings of the National Academy of Sciences*, 97(11):5699–5703, 2000. doi: 10.1073/pnas.090060097.

[12] J. Yang and F. Stern. A highly scalable massively parallel fast marching method for the eikonal equation. *ArXiv e-prints*, feb 2015.

[13] Hongkai Zhao. A fast sweeping method for eikonal equations. *Math. Comput.*, 74(250):603–627, 2005. doi: 10.1090/S0025-5718-04-01678-3.