Highway Hierarchies for HJB PDEs

Author
LIU SHANQING

Supervisor
MARIANNE AKIAN
INRIA&CMAP-Ecole Polytechnique

Co-Supervisor
STEPHANE GAUBERT
INRIA&CMAP-Ecole Polytechnique

September 16, 2020
Contents

1 Introduction .................................................. 2
  1.1 Background .................................................. 2
  1.2 The Main Goal .............................................. 2
  1.3 The General Ideal of the Algorithm ....................... 3
  1.4 The Organization of the Report ............................ 3

2 The Eikonal Equation and Fast Marching Method ............. 4
  2.1 Geometric Tools ............................................ 4
  2.2 Eikonal Equation ........................................... 5
  2.3 The Semi-Lagrangian based Fast-Marching Schemes ........ 5
  2.4 The Finite-Differences based Fast-Marching Schemes .... 6
  2.5 Difference Between the Discretization Methods .......... 7

3 The Original Highway Hierarchies Algorithm ................ 9
  3.1 Shortest Path Problem .................................... 9
  3.2 Highway Hierarchy ......................................... 10
  3.3 Computing the Highway Network .......................... 10
  3.4 Computing the Core ....................................... 11
  3.5 The Query Algorithm ...................................... 11

4 Highway Hierarchies for Eikonal Equation .................... 14
  4.1 Preliminaries .............................................. 14
  4.2 Construction of Highway Network ......................... 15
  4.3 The Semi-Lagrangian Highway Hierarchies ............... 17
  4.4 The Finite Differences Highway Hierarchies ............ 19
  4.5 Summary .................................................. 22

5 Application and Numerical Experiments ....................... 23
  5.1 Test case 1 ................................................. 23
  5.2 Numerical results for test case 1 ......................... 24

6 Conclusion ................................................................ 26
  6.1 The Summary of Current Work .............................. 26
  6.2 Proposed Future Work ...................................... 26
  6.3 Acknowledgement ........................................... 26
1 Introduction

1.1 Background

In the optimal control theorem, the dynamic programming approach is one of the major methods to find the global optimum value function and the associated feedback optimal control, and it turns the problem to find the solution of the associated fully nonlinear partial differential equations of Hamilton-Jacobi-Bellman type over the state space. Normally the computation of the optimal solution needs a time exponential in the dimension, which is so called the ”curse of dimensionally”. To find a efficiently numerical scheme for the HJB equations especially in high-dimension is one of the major topic in optimal control and algorithm fields.

Recently in computer science fields, there is a new algorithm for solving the discrete time and state shortest path problems, which is the ”Highway Hierarchies” algorithm. It is initially for applications to the on-board GPS system. In fact, given a hugh map (i.e. the road network of Western European, with more than ten millions of nodes and directed edges), after a consuming preprocessing period (i.e. around 10 mins for Western European road network), it can find the shortest path from any node to any node in a very short time (i.e. ms for the Western European road network, around thousands times faster than the general Dijkstra’s algorithm). Notice that the theoretical worst boundary is no better than the Dijkstra’s algorithm. In fact it is not a general solution, it can be efficient only for some structures of network with specific properties, i.e. large, sparse, almost planar and inherent hierarchy which means there should be inherent some ”highway” paths.

1.2 The Main Goal

Our main goal is to adapt the Highway Hierarchies method to the continuous time and state optimal control problems. The original idea is based on the relation between the discrete shortest path problem and the optimal control problems. In fact during the internship we mainly deal with the eikonal type of equation. The eikonal equation is a non-linear partial differential equation appeared in the front interface propagation problems. On of it’s physical interpretation is the ”continuous shortest-path problems”. In fact the solution of the general eikonal equation $u(x)$ could be interpreted as the minimal time in order to escape from a particular point $x$ to the boundary of the domain $\partial \Omega$.

The fast marching method, which is a numerical solution for the eikonal equation, is very similar to the Dijkstra’s algorithm in the discrete time. It can be seen as a variant of the Dijkstra’s algorithm in a discretized grids of the eikonal equation. As the essence of the
Highway Hierarchies method is to accelerate the Dijkstra’s algorithm, we would like to first adapt the ideal of highway hierarchies to the fast-marching method. And then to find the practically relevant cases in which it is efficient.

1.3 The General Ideal of the Algorithm

The highway hierarchy consists of different levels of graphs, as the level goes higher, it means the edges in the graph are "more likely" to be passed by a shortest path. Those edges are so-called the highways. So as one wants to find the shortest path from one point to another, one first do a local search of the current location, then it enters the next level of "highways" and follows the "highways", then do it iteratively. Because we reduce many edges as the level goes higher, the search space become less and less as it processes.

The ideal of the highway hierarchy should work both in discrete states and continuous states. But as in the discrete cases it requires particular properties of the network, we find that for efficiency, the eikonal equation should also have some restriction. At least there should be some inherent highways, either because the speeds are different in different position or because the physical barriers of the search spaces. We then give some suitable application cases of our algorithm.

1.4 The Organization of the Report

In the next chapter, we first introduce the context of the problem of Eikonal Equations and the existing Fast-Marching methods. Then we give a brief review of the Highway Hierarchies algorithm in the discrete cases. In chapter 4 we give the idea of our highway hierarchies for the Eikonal type of equations. In the chapter 5 we give the details of the algorithms with the numerical applications. Then we describe the main remained problems and difficulties, and the potential future work.
2 The Eikonal Equation and Fast Marching Method

The first Eikonal equation is as following:

\[ \begin{aligned}
    c(x)|\nabla T(x)| &= 1, \quad x \in \Omega \setminus \Omega_0 \\
    T(x) &= 0, \quad x \in \partial \Omega_0
\end{aligned} \]  

(1)

Where \( c(x) \) is the speed at \( x \in \Omega \). It has many applications in continuous optimal path planning, computational geometry, shape from shading, and image processing. One of it’s physical interpretation is the isotropic time-optimal control problems.

In this section we would expand the eikonal equation in more general situations with the geometric tools, and study the numerical fast-marching methods, including a finite-difference based method and the semi-lagrangian schemes. This part is mainly based on the existing work of [10] [12] [13] [14].

2.1 Geometric Tools

Let \( \mathbb{E} \) be a finite dimensional vector space, and let \( \Omega \) be a open domain of \( \mathbb{E} : \Omega \subseteq \mathbb{E} \), and equipped with closed convex sets \( \mathcal{C}(x) \), for all \( x \in \Omega \). Let \( x_0, x_1 \in \Omega \). We give the following definition of distance between \( x_0 \) and \( x_1 \):

\[ d_{\mathcal{C}}(x_0, x_1) := \min \ T, \text{ s.t. } \exists \gamma : [0, T] \to \Omega, \begin{cases} \gamma(0) = x_0, \gamma(T) = x_1 \\ \gamma'(t) \in \mathcal{C}(\gamma(t)), \forall t \end{cases} \]  

(2)

Here the points are the elements in \( \Omega \), a path is a map \( \gamma \in C([0, 1], \Omega) \) and \( \mathcal{C}(x) \) is a control set for \( x \in \Omega \). The path length is measured using a given norm \( \mathcal{F}_x \) associated with a control set \( \mathcal{C}(x) \), \( \mathcal{F}_x(v) \) is the Minkowski distance:

\[ \mathcal{F}_x(v) := \min \{ \lambda \geq 0 ; v \in \lambda \mathcal{C}(x) \} . \]  

(3)

Here are various possible expressions of the metric:

- Isotropic: \( \mathcal{F}_x(v) = c(x) \| v \| \), with \( c(x) > 0 \).
- Riemannian: \( F_x(v) = \sqrt{v^T M(x) v} \), with \( M(x) \in S^+_d \).
- Rand: \( F_x(v) = \sqrt{v^T M(x) v + \langle w(x), v \rangle} , w(x) \in \mathbb{R}^d \).
- Non-holonomic: \( F_x(v) = +\infty \) for some \( v \).
Then the distance from $x_0$ to $x_1$, associated to the metric $F_x$, could be presented as:

$$d_F(x_0, x_1) = \inf_{\gamma} \int_{0}^{1} F_{\gamma(t)}(\gamma'(t))dt.$$  

subject to $\gamma(0) = x_0$, $\gamma(1) = x_1$ and $\gamma \in Lip([0, 1], \Omega)$.

### 2.2 Eikonal Equation

Here in our optimal control problem, we intend to find the shortest path from the boundary of a domain $\Omega$ to a given point $x \in \Omega$. We first define the distance map $u(p)$ from a point $p_0 \in \partial \Omega$ with the definition of distance and path:

$$u(x) := \inf \{ \int_{0}^{1} F_{\gamma(t)}(\gamma'(t))dt; \gamma : [0, 1] \rightarrow \Omega, \gamma(0) \in \partial \Omega, \gamma(1) = x \}$$

The function $u(x) : \Omega \rightarrow \mathbb{R}$ is known to be the viscosity solution to the generalized eikonal equation:

$$F^*_x(\nabla u(x)) = 1, \quad F^*_x(w) := \max\{ \langle v, w \rangle; F_x(v) \leq 1 \}$$

for all $x \in \Omega$ with boundary condition $u(x_0) = 0$ for all $x_0 \in \partial \Omega$. The value function $u(x)$ can be seen as the arrival time at $x \in \Omega$ from the boundary $\partial \Omega$ propagating at speed w.r.t. the metric $F$.

Notice that the equation(5) is to find the value function associated to the point $x \in \Omega$. Thus, in order to find the shortest geodesic, we need to further solve the ODE:

$$\gamma'(t) = V(\gamma(t)), \forall t \in [0, 1]$$

where $V(x) := \nabla F^*_x(\nabla u(x)), \forall x \in \Omega$, with $\gamma(0) \in \partial \Omega, \gamma(1) = x$. This is based on the observation that geodesics follow the direction of greatest slope of the value function w.r.t. the metric $F$.

### 2.3 The Semi-Lagrangian based Fast-Marching Schemes

The Semi-Lagrangian schemes for the eikonal equation are based on the Bellman’s optimality principal characterization for the shortest path. By the Bellman’s optimality principle, a shortest path from the boundary $\partial \Omega$ to some point $x \in \Omega$ should be the concatenation of two shortest path: first from the boundary $\partial \Omega$ to some $y \in \partial V(x)$ where $V(x) \subseteq \Omega$ is an arbitrary but fixed neighbourhood of $x$, then from $y$ to $x$. Hence by the previous definition we have:

$$u(x) = \inf_{y \in \partial V(x)} d_F(x, y) + u(y).$$

Numerically the semi-lagrangian scheme based on this principle in the discrete setting. We use finite sets $X, \partial X$ approximating $\Omega$ and $\partial \Omega$. For all $x \in X$ let $V(x)$ be a polytope neighbourhood enclosing $x$ and which vertices belong to $X \cup \partial X$. Then the semi-lagrangian discretization is finding $U : X \cup \partial X \rightarrow [0, \infty]$ as an approximation of the value function $u(x)$, such that for all $x \in X$:

$$U(x) = \min_{y \in \partial V(x)} F_x(y - x) + I_{V(x)}U(y).$$
With boundary $U(x) = 0$ for all $x \in \partial X$. Notice that $I_{V(x)}$ is a linear interpolation operator, since the optimal point in the neighborhood $V(x)$ could be obtained between two vertices. Then the system (8) approximate system (7) by replacing the distance function with the local gauge of the metric, and approximate the value function $u(x)$ by the piecewise linear interpolation $I_{V(x)}U(y)$ of the approximation $U$.

If we further define the operator $\wedge_{SL}$ as:

$$\wedge_{SL}U(x) := \min_{y \in \partial V(x)} \left( F_x(y - x) + I_{V(x)}U(y) \right).$$

(10)

Then we need to solve the fixed point problem: $\wedge_{SL}U = U$. This operator is by construction monotone, this property implies the convergence of iterative schemes for solving the system (8).

In order to solve (8) by the fast-marching method, we need an additional property called causality: An operator $\wedge : \mathcal{U} \to \mathcal{U}$ is said to be causal is for all $U, V \in \mathcal{U}$ and all $\lambda \in [-\infty, +\infty]$ one has $U < \lambda = V < \lambda \Rightarrow (\wedge U) < \lambda = (\wedge V) < \lambda$.

The acuteness implies causality: The operator $\wedge$ is causal if for any $x + u, x + v$ in a common facet of $V(x)$: $(\nabla F_x(u), v) > 0$.

The discrete system then can be solved in a single pass using a variant of Dijkstra’s algorithm, here we give a sketch of the Semi-Lagrangian Fast Marching algorithm.

Algorithm 1 Semi-Lagrangian Fast Marching

Input: An operator $\wedge$, Monotone and Causal

Output: $u$, which is now a fixed point of $\wedge$

Initialization: set $u \leftarrow \wedge(+\infty)$, tag all points as non-accepted.

1: while there remains a non-accepted point do
2: Let $x \in X$ be a minimizer of $u$ among non-accepted points;
3: Tag x as accepted;
4: for $\forall y$ non-accepted and such that and s.t. $\wedge u(y)$ depends on $x$ do
5: $u(y) \leftarrow \wedge u(y)$
6: end for
7: end while

2.4 The Finite-Differences based Fast-Marching Schemes

In this part we introduce the finite-differences based methods. Although the finite-differences method is hardly interpretable, but contrary to the semi-lagrangian scheme which needs a complex geometrical computations (this causes many difficulties in our numerical experiments), the finite-differences methods based on a direct upwind discretization of the eikonal PDE, makes the implementation be straightforward.

We first need to define the Lagrangian and Hamiltonian associated to the metric $\mathcal{F}$: For any $x \in \Omega$, and any vector $w \in E$ and co-vector $v \in E_d^*$, we have:

$$\mathcal{L}_x(w) := \frac{1}{2} F_x(w)^2, \quad \mathcal{H}_x(v) := \sup_{w \in E_d} \langle v, w \rangle - \mathcal{L}_x(w).$$

(11)
We can check that for Riemannian metric, the Hamiltonian is the half squared dual metric, i.e. $\mathcal{H}_x = \frac{1}{2}(\mathcal{F}_x)^2$. Then the eikonal equation (5) can be rewritten in terms of Hamiltonian:

$$2\mathcal{H}_x(\nabla u(x)) = 1, \forall x \in \Omega.$$  \hfill (12)

where $u(x) = 0, \forall x \in \partial \Omega$. Then discretization of the PDE (11) rely on an approximation of the Hamiltonian $H_x(w) \approx \mathcal{H}_x(w)$ with the following:

$$2H_x(w) = \max_{i \in I} \sum_{j \in J} \rho_{i,j}(x) (\alpha_{i,j}(x))_+^2.\hfill (13)$$

where $I$ and $J$ are finite sets, and the choice of weights $\rho_{i,j}(x)$ and offsets $\alpha_{i,j}(x)$ depends on the structure of the metric. The discretization of the eikonal PDE can be written as following: find $U : X \cup \partial X \rightarrow \mathbb{R}$, such that:

$$\max_{i \in I} \sum_{j \in J} \rho_{i,j}(x)(U(x) - U(x - h\alpha_{i,j}(x)))_+^2 = h^2, \forall x \in X.$$ \hfill (14)

where $U(x) = 0, \forall x \in \partial X$. The left hand side of the equation (13) is monotone and causal, since $(\cdot)_+$ means the positive parts of the finite differences i.e. $(U(x) - U(y))_+ = \max\{0, U(x) - U(y)\}, x, y \in X$. Thus it could be solved by the fast-marching algorithm.

Also in order to apply the fast-marching method, we define an operator $\wedge_{FD}$ for the finite differences scheme. We first define a function $\varphi : X \times \mathbb{R} \times \mathbb{R}^X \rightarrow \mathbb{R}$, for all $x \in X$ and all $U \in \mathbb{R}^X$:

$$(\varphi U)(x) := \varphi(x, U(x), (U(x) - U(y))_{y \in X}).$$ \hfill (15)

Then we can reformulate the previous system $\varphi U = 0$ by a fixed point operator $\wedge_{FD} U = U$. We define the operator $\wedge_{FD} : U \rightarrow U$, for all $x \in X$ and $U \in U := [-\infty, +\infty]^X$:

$$\wedge_{FD} U(x) := \sup\{V(x) ; V \in \mathbb{R}^X, (\varphi V)(x) \leq 0, V \leq U \text{ on } X \backslash \{x\}\}.\hfill (16)$$

We recall for 2.3 the definition of causality. And the system can be solved by the same algorithm in Algorithm 1 Semi-Lagrangian Fast Marching, only change the operator form $\wedge_{SL}$ to $\wedge_{FD}$.

Here we give more details about the upwind discretization, if we denote $e_{i,j} = \alpha_{i,j}(x)$, for all $x \in X$, the left hand side of equation (13) could be discretized as following:

$$\mu_{i,j}(x) \max\{0, U(x) - U(x - he_{i,j}), U(x) - U(x + he_{i,j})\}^2, \mu_{i,j}(x) (w, e_{i,j}(x))^2.\hfill (17)$$

The numerical schemes should vary from different models and structure of the metric, but the corresponding scheme should based on the stencils which is the convex hull of $\{x \pm he_{i,j}\}$.

### 2.5 Difference Between the Discretization Methods

In this section, we give a sample analysis of the difference and connection between these two methods.

First of all, the semi-lagrangian discretization method is based on the Bellman’s optimality principle, which makes it more understandable, and it is more trivial from geometric point of view. The finite difference scheme relies on a direct discretization of the eikonal equation,
which needs a further analysis of the property of Hamiltonian and lagrangian, and the associated metric, which is more hardly interpretable.

However the semi-lagrangian scheme needs a complex geometric computation, which brings some difficulties in the numerical implementation. In contrary the finite-differences scheme is more straightforward in the numerical application, and is more similar to the shortest path problem in discrete case.

Another advantage of semi-lagrangian scheme is that it can be easily extended to meshed domains, but the finite difference scheme normally needs a cartesian grid.
3 The Original Highway Hierarchies Algorithm

In this section we introduce the original highway hierarchies algorithm for the discrete shortest path problem. The Highway Hierarchies method is a speedup technique based on the Dijkstra’s algorithm for the route planning problem in real world transportation networks. It is initially for the applications to on board GPS systems.

The basic idea for the highway hierarchies method is that: just do a ”local search” around the source and target nodes, when we get far away from the source and target only ”important roads” are considered. A highway hierarchy of a graph $G$ consists several levels $G_0, G_1, ..., G_L$. $G_0$ is the original graph, each level graph $G_l$ is obtained by computing the highway network of the core of $G_{l-1}$. More details about the existing work could be found in [4] [17].

Many other query algorithms could be applied based on the highway hierarchies, in order to further accelerate the algorithm. For example the $A^*$ search using landmarks (ALT). More details about the query methods based on the highway structure can be found in [15] [5] [8].

3.1 Shortest Path Problem

Dijkstra’s Algorithm. The Dijkstra’s algorithm is proposed by the computer scientist Dijkstra in 1956, its original idea is to find a shortest path between two given source and target nodes. It’s one of the basic method in algorithms and has many variants, that to find the shortest paths from one source node to all other nodes and to find the shortest path from every node to every node.

We denoted by $G = (V, E)$ a directed graph with $n$ nodes $u \in V$, and edges $(u, v) \in E$, each edge has an associated weight $w(u, v)$. The Dijkstra’s algorithm start searching from a single source node $s \in V$, and growing a shortest path tree in order to find all the shortest paths from other node to the start node $s$ and denoted by $d_s(v)$. During the search, we give every node in the graph a state $unreached$, $reached$, $settled$. At the start of the search, we set the source node as $reached$, and all other nodes are $unreached$, i.e $d_s(v) = +\infty$. At each iterative select the closed node in the neighborhood of current $reached$ nodes, update the distance table $d_s(v)$ and set it as $reached$. When we have done done considering all the neighborhoods of current $reached$ node, we set it as $settled$, the distance $d_s(v)$ of $settled$ nodes will never change and the shortest path has been found.

Another notation is the Dijkstra rank. If we do a Dijkstra’s search starting from $s$, then all the nodes with state $settled$ are settled in a specific order. We called this order $rk_s(u)$, for all $u$ in the $settled$ set of $s$. 
3.2 Highway Hierarchy

The highway hierarchy consists of several levels of graph : $G_0, G_1, ... G_L$, in which $G_0$ is the original graph, and each level $G_{l+1}$ graph is generated from $G_l$.

In order to construct the highway hierarchy, we need a further definition of the highway neighborhood: $N^l(v)$. It is defined by a parameter $H^l \in \mathbb{R}^{V_l}$. For every $v \in V_l$, the highway neighborhood $H_l(v)$ is defined as following: Do a general Dijkstra’s algorithm started from $v$, then we will get a shortest path tree, $N^l(v)$ contains all the first $H^l(v)$ nodes the shortest path tree pass through.

The construction of the highway hierarchy normally contains two steps. The first step is constructing the highway network, from $G_l'$ to $G_{l+1}$, in this step we do the edges reductions. The highway network $G_{l+1} = (V_{l+1}, E_{l+1})$ is defined as following: every highway edge $(u,v)_{l+1} \in E_{l+1}$ should satisfy two conditions:

- there exists two nodes $s'_l \in V_l'$ and $t'_l \in V_l'$ such that $(u,v)_{l+1}$ appears in some shortest path $<s,...,u,v,...,t>$.  
- $(u,v)_{l+1}$ is not fully contained in the highway neighborhood of $s'_l$ and $t'_l$, i.e. $v_{l+1} \notin N^l\rightarrow(s'_l)$ and $u_{l+1} \notin N^l\leftarrow(t'_l)$.

The node set $V_{l+1}$ is the set of nodes in $V_l'$ associated with some edge in $E_{l+1}$.

The second step is the construction of the core, which is the step of nodes reduction. In this step we have a set $B_l \subseteq V_l$ of the bypassable nodes. We then can test if those nodes could be bypassed by introduced a set of shortcuts $S_l$. After these bypass process, the core $G_l' = (V_l', E_l')$ of $G_l$ consists as follows: $V_l' := V_l \setminus B_l$ and $E_l' := (E_l \cap (V_l' \times V_l')) \cup S_l$.

Notice that we need the next level’s highway structure contains all the highway edges of the previous level, then combine with an appropriate query algorithm, we will get the exact shortest path based on the highway structure, not the approximated one. Also notice that the preliminaries of the nodes reduction needs the structure of the graph to be also planar, and the parameter of the test for bypassability will control the contraction rate from level to level. In the next sections we will give more details about the contraction process.

3.3 Computing the Highway Network

We want to construct the higher level graph $G_{l+1} = (V_{l+1}, E_{l+1})$ based on the current $G_l' = (V_l', E_l')$. Following we give the details of the construction algorithm but without theoretical proofs.

We start with an empty set $E_{l+1}$, for every node $s'_l \in V_l'$ we do a partial shortest-path Dijkstra’s algorithm:

- The source node $s'_l$ is setted as active, a shortest path tree from $s'_l$ is constructed by the Dijkstra’s algorithm.
- During the search, every node reached for the first time and every updated reached node is setted as active if any of it’s tentative parents is active.
- When a node $p$ is settled, we check whether $p$ is passive by all the shortest paths $P'$ from $s'_l$. 


when no active unsettled node is left, the search is aborted and the growth of the shortest path tree stops.

The notation reached, settled and tentative parents are the same as normal Dijkstra’s algorithm. The state of a node \( p \) is passive iff: Every shortest path \( P' \) from \( s'_l \) to \( p \): \( P' = \langle s'_l, ..., p \rangle \), we have: \( s'_l < p \land p \notin N^l\rightarrow(s'_l) \land s'_l \notin N^l\leftarrow(p) \land |P' \cap N^l\rightarrow(s'_l) \cap N^l\leftarrow(p)| \leq 1 \), where \( s'_l \) is the first successor of \( s'_l \).

The abort condition of passive actually means that, if we reach a passive node \( p \), we can ignore everythings lies behind \( p \), beacuse it ensures that further important edges will be added during the search started from \( s'_1 \) (The same situation in the search from \( s'_1 \), it’s first successer will gets further important edges and so on).

After completing the search for every nodes in \( G'_l \), we then collect all the edges \((u, v)\) such that: it lies on some shortest path from a source node \( s'_l \) to a passive node \( p: \langle s'_l, ..., u, v, ..., p \rangle \) and \( v \notin N^l\rightarrow(s'_l) \), \( u \notin N^l\leftarrow(p) \), into the next level \( E'^l+1 \).

3.4 Computing the Core

After the step of edges reduction, we further do a step of nodes reduction. The idea is simple: we check if we could bypass some nodes by introducing some shortcuts, if we do so the structure of the graph become simpler then we bypass those nodes.

Here we give a iterative algorithm of both selection of the bypass nodes and creation of the shortcuts. First we let all nodes as considered, and we start with the topmost node \( u \), the criterion of the bypassability is as follows: \#shortcuts \( \leq c \cdot (deg_{in}(u) + deg_{out}(u)) \). The constracting rate is depended on the parameter \( c \). It the criterion is fulfilled we the node can be bypassed. Notice that: If \( c < 2 \), then \(|E'_l| = O(|V_l| + |E_l|)\).

3.5 The Query Algorithm

The original query algorithm based on highway hierarchies is a modification of the bidirectional version of Dijkstra’s algorithm, and the forward and backward are symmetric.

We start with a source node \( s_0 \) is the original level 0 graph. First do local search in the level-0 neighborhood of \( s_0 \) by the dijkstra’s algorithm. If an edge \((u, v)\) crosses the neighborhood border, then we switch to the higher search level \( l \). If the level of edge \((u, v)\) is less than the new search level we also keep it in the new level. Then \( v \) adapted the new level of search \( l \). Also notice there are several special cases in the query, but those situations will not appear in our further algorithms so we don’t give the details.

The query algorithm mentioned above is the original and simplest search method based on highway hierarchies, but recently some reseachers working on shortest path algorithms are trying to combine some advanced query methods with the highway hierarchies structure. since those query algorithms seems really interesting an related to our PDE aprochesm we give a short reviews of them.

**the Goal-Directed Search.** The general idear of the goal-directed search is to ”guide” the search toward the target, in case by avoiding some vertices that are not in the direction to the target. One of the famous method is \( A^* \) search.

**\( A^* \) search:** Let \( d(u, v) \) denote the distance between node \( u \) and \( v \). The general idea of \( A^* \) search is to build a potential function: \( \pi : V \rightarrow \mathbb{R} \), such that for every node \( v \in V \), \( \pi(v) \) will be the lower bound of \( d(v, t) \), i.e. the distance between \( v \) and the target node. By
doing so, when do the normal Dijkstra search, we will calculate \( d(s, u) + \pi(u) \). Then the vertices that are closer to the target will have more possibilities to be reached. In fact if the potential function is the exact lower bound, i.e. \( \pi(v) = d(u, t) \), then only the nodes in the true shortest path will be passed.

The \( A^* \) search can be a bidirectional method. An efficient way to decide the backward search and forward search potential functions is to combine two feasible potential functions \( \pi_f \) and \( \pi_b \). Use \( \pi_f - \pi_b \) as the forward potential function, and \( \pi_b - \pi_f \) as the backward potential function.

\[ \text{ALT Search.} \quad \text{The} \quad \text{"quality" \quad of \quad the \quad potential \quad function \quad determines \quad the \quad performance \quad of \quad the \quad goal-directer \quad search. \quad One \quad efficient \quad way \quad to \quad define \quad the \quad potential \quad function \quad is \quad the \quad ALT \quad algorithm (A^* landmarks, and triangle inequality):} \]

First do a pre-process, select a small set \( L \subseteq V \) as the landmarks nodes and calculate the distance between them and all the nodes in the graph. Then during the search from two particualr nodes \( s, t \), there is a triangle inequalities: for every \( l_i \in L \):

\[
\begin{align*}
  d(u, t) &\geq d(u, l_i) - d(t, l_i) \\
  d(u, t) &\geq d(l_i, t) - d(l_i, u)
\end{align*}
\]  

(18)

These triangle inequalities give a valid lower bound of the distance \( d(u, t) \), and the lower bound could take the maximum over all the landmark nodes. In the normal graph, we usually take the well-spaced nodes in the boundary as the landmark nodes.

\[ \text{Precomputed Cluster Distances.} \quad \text{Another useful goal-directed search methods is the Precomputed Cluster Distances (PCD) method.} \quad \text{It based on a pre-partition of clusters (} C_1, ..., C_K \text{)}, \quad \text{the pre-process compute the shortest paths between different clusters. After this, when we do the normal Dijkstra search, the lower bound of the distance to the target node from the reached node can be written as:} \]

\[ d(s, u) + d(C(u), C(t)) + d(v, t), \text{where} \quad C(u) \quad \text{is the cluster contains} \quad u, v \quad \text{in the boundary of} \quad C(t) \text{. This method requires less space than the ALT search method.} \]

In the following we give a sketch of the Highway Hierarchies algorithm based on normal query method:

### Algorithm 2 Highway Hierarchies Search

**Input:** the source node \( s \), the target node \( t \)

**Output:** the distance from \( s \) to \( t : d(s, t) \)

**Initialization:** \( d' := \infty \), insert \((Q^\rightarrow, s, (0, 0, r_0^\rightarrow(s)))\), \((Q^\leftarrow, t, (0, 0, r_0^\leftarrow(t)))\).

1. while \((Q^\rightarrow \cup Q^\leftarrow \neq \emptyset)\) do
2. select the forward or backward direction s.t. \( Q \neq \emptyset \); 
3. let \( u := \text{deleteMin}(Q) \); 
4. if \( u \) is settled in both directions then 
5. \( d' = \min\{d', \sigma^\rightarrow(u) + \sigma^\leftarrow(u)\} \)
6. end if 
7. if \( \text{gap}(u) \neq \infty \) then 
8. \( \text{gap} := \text{gap}(u) \); 
9. else 
10. \( \text{gap} := r_l(u) \); 

12
11: end if
12: for $\forall e = (u,v) \in E$ do
13: for $l := l(u), gap := gap', w(e) > gap$ do
14: $l + +, gap := ri(u);$ 
15: end for 
16: if $l(e) < l$ then continue 
17: end if 
18: if then $u \in V' \land v \in B_l$ continue 
19: end if 
20: $k := (\sigma(u) + w(e), l, gap - w(e));$
21: if $v$ has not been reached then 
22: insert $(Q, v, k);$ 
23: end if 
24: end for 
25: end while
4 Highway Hierarchies for Eikonal Equation

In this section we will describe how to adapt the highway hierarchies method to solve the eikonal equations. Recall that from chapter 2, the fast-marching method is very similar to the Dijkstra’s algorithm as long as we get the fix-pointed operator $\wedge$. This operator could be obtained by various schemes of discretization methods. Thus we also first aim to construct a ”highway hierarchies” of the discretization space, after that we then search in this structure instead of the original discretization space.

Also notice that the fast-marching method is mainly for computing the value function of the eikonal equation, which corresponds to the distance in shortest path problem. But the highway hierarchies method needs to extract the exact shortest path in every step and every iteration of the algorithm, which brings difficulties especially for the semi-lagrangian scheme. Another main difficulty is that, normally to make sure the highway hierarchies method to be efficient we need the graph to have some particular properties, i.e. the graph should be large enough, almost planar and there should be some inherent hierarchy of important paths. This is not trivial for the general discretized space of eikonal equation.

4.1 Preliminaries

Let $\Omega$ be an open domain of $E : \Omega \subseteq E$, and let the finite sets $X$ and $\partial X$ be the approximations for $\Omega$ and $\partial \Omega$. We first give the definition of the Highway Neighborhood. Let $H := \{H^1, H^2, ..., H^L\} \subseteq \mathbb{R}^L$ be a given parameters for the highway hierarchy of the discretization grid $X \cup \partial X$. For every $x^l_i$ in $X^l \cup \partial X^l$, we first do a fast marching search (Algorithms 1, for both the schemes) started from $x^l_i$, then we will get a set of points $Y^l_i = \{y^i_{1l}, y^i_{2l}, ..., y^i_{H^l_l}\}$ that the fast marching has gone through, i.e. for every $y^i_{jl} \in Y^l_i$ the value function $U(y^i_{jl})$ is determined during the search and will not change afterwards. The set has a specific order by the order that $y^i_{jl}$ has been settled by the fast-marching search. Then the highway neighborhood for $x^l_i$ in the level $- l$ of the highway structure will be the first $H^l$ nodes that added to the set $Y^l_i$.

**Definition 1.** The set $N^l(x^l_i) = \{y^i_{1l}, y^i_{2l}, ..., y^i_{H^l_l}\}$ is called the highway neighborhood of $x^l_i \in (X^l \cup \partial X^l)$ in level $l$, and $r^l(x^l_i) = U(y^i_{H^l_l})$ is called the neighborhood radii.

In fact the definition of the neighborhood means ”$H^l$ closest points” of the point $x^l_i$ in the fast-marching search, it corresponds to the ”local search” part. The choice of the parameter $H$ could influence the construct of the highway network.
After we need to define the highway. The idea of the highway hierarchy for the fast-marching method is the same as in discrete case: First do a normal local search until we find some "highways", then we will follow the highway to further search. The reason that this method can accelerate the normal fast-marching after the pre-process is when you are searching in the highway, the needed calculation of the approximation value functions is reduced, since the higher level highway structure contains less edges than the previous one and thus the neighborhood nodes of current front interface are less. Then at every iteration i.e. when we update \( U(y) \) by the operator \( \land U(y) \), there are less nodes we should to approximate the value function and decide the next search point.

Let \((x_i, x_j)\) be an edge in the discretization grid \(X \cup \partial X\), then:

**Definition 2.** The edge \((x_i, x_j)\) is a highway edge if the following conditions are fullfilled:

- \((x_i, x_j)\) belongs to some shortest path from \(x_s \in X \cup \partial X\) to \(x_t \in X \cup \partial X\),
- \(x_i \notin N(x_0)\) and \(x_j \notin N(x_t)\).

The highway edges should consist of all the "important paths", in which the shortest paths have more chances to pass through. So obviously they should be shown in some shortest paths, which is the first property.

The second property means that the highway should be far away enough from the source and target points, such that after a proper local search in the neighborhood of source point, one could find the good highway and go through it. This property purpose to find the "common highways" for many different positions. In fact we could find later that the algorithm is more efficient if there are more "common highways" shared by different positions. Otherwise it would cost too much to find the highway.

The aim of the construction of highway hierarchy is to find all the possible highway edges in the current level, and then connect them with all the nodes to the next level's network. This also makes sure that what we find the by search from level to level is the exact shortest path as we can find in normal fast-marching. The highway structure contains different levels, and the definition of level is trivial as following:

**Definition 3.** The level of a node \(l(x_i)\) and an edge \(l((x_i, x_j))\) is the level of the highest highway network in which they belong to.

It's obviously that as the level goes higher, there are greater possibilities it can be passed through by a shortest path. The notations of level counts when we finish a local search and to decide whether in the next step we should go into the next level or not.

### 4.2 Construction of Highway Network

The construction of the highway structure is the pre-process phase of our algorithm, and it is also the most consuming phase. Our inherent purpose is to reduce the search space as it progresses. We intend to construct levels of discretization spaces \((X^l \cup \partial X^l)\) to reduce the edges.

To construct the highway network of the discretized space, a first pre-process is needed to get the highway neighborhood sets of the every point in the discretized grids of the current level as we mentioned before, we denote by \(N^l(x^l), \forall x^l \in (X^l \cup \partial X^l)\).
Then to complete the next-level’s highway structure based on current level, we do a
two-phases search: For every node \( s \in (X^l \cup \partial X^l) \) in the current-level \( l \) grid,

- The first phase is to perform a normal fast-marching search started from \( s \) till the abort
  condition is satisfied;
- The second phases is to select the proper edges into the next level’s highway network.

**Phase 1:** For every node \( x^l_i \in (X^l \cup \partial X^l) \), we perform a partial fast-marching search,
during the search we give every node that have been passed a state ”active” or ”passive”.
We first set \( x^l_i \) to active; as the propagation progress, we get a set of nodes have been
reached \( Y^l_i = \{y^i_{1,1}, y^i_{2,1}, \ldots\} \) in which every node \( y \in Y^l_i \), the value function \( u(y^i_{1,1}) \) has been
approximated by the fixed point operator \( \land \), then we could use the value function to extract
the shortest path \( P_{x^l_i,y^i_{1,1}} \) from \( x^l_i \) to every \( y^i_{1,1} \in Y^l_i \). We first mark all these nodes that are
first added in \( Y^l_i \) as active, the state of a node is changed to passive if the following property
is satisfied:

**Definition 4.** A node \( y^* \) is called passive in the shortest path tree started from \( x^l_i \), if the
following equation satisfied: \( |P_{x^l_i,y^*} \cap N^l(x^l_{i,1}) \cap N^l(y^*)| \leq 1 \)

Where \( P_{x^l_i,y^*} \) denote some particular shortest path from \( x^l_i \) to \( y^* \). \( x^l_{i,1} \) is the first successer
of \( x^l_i \) in \( P_{x^l_i,y^*} \). If the above equation is satisfied we let \( y^* \) to be passive.

Notice that during the fast-marching search, we give each point another state called
unreached, reached, settled as the same notation as in Dijkstra’s algorithm (check in 3.1).
Briefly speaking, the point \( z \) has not been touched i.e. \( U(z) = +\infty \) is unreached; the point
\( z \) has been touched and it’s value function \( U(z) \) has been approximated by the fast-marching
operator is reached; the point \( z \) which all it’s dependent neighborhood ponit’s value function
has been calculated and thus it’s value function \( U(z) \) is determined and will never change as
the propagation process is called settled.

After that the abort condition is as following:

**Definition 5.** The abort condition: No active unsettled node is left.

In fact, the abort condition means there is at most one node in the intersection between
the neighborhood \( N^l(x^l_i) \) and \( N^l(y^*) \) in shortest path from \( x^l_i \) to the passive node \( y^i_{j,l} \). When
\( x^l_{i,1} \) and \( y^i_{j,l} \) are close, in the case the propagation just begin, their neighborhood will have
many nodes in common. As the search progresses, they will have less and less nodes in
common. When every node in the front interface has less than one nodes in common with
the source node on their neighborhood we abort the search, in which means that the interface
has propagated far away enough from the original node. And thus all the nodes outside the
current front interface are passive by the previous definition of passive.

**Phase 2:** After phase 1, for every node \( x^l_i \in (X^l \cup \partial X^l) \) we will get a set of passive
nodes \( Y^*_{i,l} = \{y^i_{1,1}, y^i_{2,1}, \ldots, y^i_{j,k}\} \), every node in \( Y^*_{i,l} \) is the first settled node in their direction
of shortest path that changed from active to passive. All these nodes also contrut a front
interface centered with the start point \( x^l_i \). Every node \( y^* \in Y^*_{i,l} \) have the following perporties:

- \( y^* \) is stated passive for \( x^l_i \), every node in \( (X^l \cup \partial X^l) \) outside the front interface formed
  by \( Y^*_{i,l} \) is satisfied the passive condition for \( x^l_i \).
The next level’s discretized space (Phase1 and Phase2, all the highway edges in \(X^t\)) contain all the nodes of \(X^t\) and formed them. We admit two kinds of fast-marching methods based on different discretization methods: Semi-Lagrangian discretization and finite differences discretization. The general idea of the pre-process is always do the local search to find the proper highway edg. and formed them. Using the same geometry tools as in section 2.1 and 2.2, we first recall the eikonal equation from the continuous time point of view:

\[ F_x^*(\nabla u(x)) = 1, \quad F_x^*(w) := \max\{\langle v, w \rangle ; F_x \leq 1\} \]  

(19)

where \(x \in \Omega\), and \(w\) is a vector \(w \in \mathbb{E}\) with the co-vector \(v \in \mathbb{E}^*_d\), the boundary condition is \(u(x_0) = 0, \forall x_0 \in \partial \Omega\).

The semi-Lagrangian discretization is based on the Bellman’s optimality principle: the optimal path from the boundary to \(x\) should be the optimal path from the boundary to the main result:

**Theorem 1.** By Phase1 and Phase2, all the highway edges in \((X^t \cup \partial X^t)\) are added into the next level’s discretized space \((X^{t+1} \cup \partial X^{t+1})\).

**Proof.** We recall the definition of highway edges (definition 2). Suppose that there exist \(s, t\) in the current level and the shortest path from \(s\) to \(t\) we denoted by \(P_{s,t}\). Suppose there exist \((u, v) \in P_{s,t}\) and \(u \notin N^t(s)\), \(v \notin N^t(s)\). We have to prove that after phase1 and phase2, \((u, v)\) is added to the next level.

First, if \(t \notin Y^*_s\), i.e. \(t\) has been passed through by the search started from \(s\) in phase 1, then it is trivial that \((u, v)\) is selected into \((X^{t+1} \cup \partial X^{t+1})\) by phase 2.

If \(t \notin Y^*_s\), i.e it is outside the interface formed by \(Y^*_s\), we prove by contradiction. Because \(v \notin N^t(s)\), there exist \(s_0 \in P_{s,t}\) such that \(v \notin N^t(s_0)\) and \(v \notin N^t(s_1)\), where \(s_1\) is the first successor of \(s_0\) in \(P_{s,t}\). We denote \(p \in P_{s_0,t}\) the first node changed from active to passive during the search started from \(s_0\). Then we have \(|P_{s_0,p} \cap N^t(s_1) \cap N^t(p)| \leq 1\). Because \(p \notin N^t(s_1)\) and \(v \in N^t(s_1)\) we have that \(v\) is closer to \(s\) than \(p\). Suppose that \(u \in N^t(p)\), scince \(s_0 \notin N^t(p)\), \(s_0\) is closer to \(s\) that \(u\) and so does \(s_1\) closer or equal to \(u\).

By the similar analysis, we have \(v \in V^t(p)\). So we have \((u, v) \subseteq P_{s_0,p} \cap N^t(s_1) \cap N^t(p)\), and \(|P_{s_0,p}| \cap N^t(p) \geq 2\). Contradiction.

The different discretization methods make Phase 1 search varies from case to case, but the general idea of the pre-process is always do the local search to find the proper highway edgs and formed them. We admit two kinds of fast-marching methods based on different discretization method: Semi-Lagrangian discretization and finite differences discretization. In the following sections we will give the details of Semi-Lagrangian highway hierarchies algorithm and Finite-Difference highway hierarchies algorithm.

### 4.3 The Semi-Lagrangian Highway Hierarchies

In this section we present the complete Semi-Lagrangian Highway Hierarchies scheme. Using the same geometry tools as in section 2.1 and 2.2, we first recall the eikonal equation from the continuous time point of view:

\[ F_x^*(\nabla u(x)) = 1, \quad F_x^*(w) := \max\{\langle v, w \rangle ; F_x \leq 1\} \]  

(19)

where \(x \in \Omega\), and \(w\) is a vector \(w \in \mathbb{E}\) with the co-vector \(v \in \mathbb{E}^*_d\), the boundary condition is \(u(x_0) = 0, \forall x_0 \in \partial \Omega\).
neighborhood of \( x \) plus the path from the neighborhood to \( x \). For every \( x \in \Omega \), for any neighborhood \( V \) of \( x \) in \( \Omega \), this property can be written as:

\[
u(x) = \inf_{y \in \partial V} (d_F(x, y) + u(y)).\tag{20}
\]

In order to numerically calculate \( u(x) \) in discretized space, we use finite sets \( X \) and \( \partial X \) to approximating \( \Omega \) and \( \partial \Omega \), and using a polyhedral neighborhood \( V(x) \) to approximate the neighborhood \( V \) of each \( x \in X \cup \partial X \) with vertices in \( X \cup \partial X \). The fix point update operator for fast-marching methods then can be denoted by:

\[
\wedge_{SL} U(x) := \min_{y \in \partial V(x)} (F_x(y - x) + I_{V(x)} U(y)).
\tag{21}
\]

\( I_{V(x)} \) is a linear interpolation operator because the optimal point \( y^* \) may not be in the vertices of our discretization grids, thus it’s approximated value function \( U(y^*) \) should also be approximated by the piecewise linear interpolation of the \( U(y), y \in \partial V(x) \). It is not a problem for computing the value function, but it needs a further attempt to get the exact shortest path.

However this linear interpolation approximation causes the main difficulty when constructing the highway hierarchy, because this approximation makes the shortest path do not go through the discretization grid. But when we construct the highway network, it’s based on the current discretized grids, and select some edges in some shortest paths into the next level, it’s difficult to choose which grid should added to next level when the shortest path do not go through the edges.

We consider a way to overcome this difficulty, and construct the highway structure based on the semi-lagrangian discretization, but just for the level-1 hierarchy of the original one. The higher levels construction needs a further analysis for the performance of the Equation in the high-level discretization grid.

**Highway structure:** For each node \( x_i \), we first do the phase 1 search by the fast-marhing operator \( \wedge_{SL} \) and get the set of nodes \( Y_i = \{y_1, y_2, \ldots\} \) have been passed through. After we leave the highway neighborhood \( N(x_i) \), denoted the leave node by \( y_l \), we start to extract the shortest path from \( x_i \) to every node added to \( Y_i \) by computing the geodesic flow using the following method:

**The geodesic flow:** The geodesic \( \gamma_{x,y} \) from a node \( x \) to \( y \) obeys the following ordinary differential equation:

\[
\gamma'(t) = V(\gamma(t)) \tag{22}
\]

With boundary conditions \( \gamma(0) = x \) and \( \gamma(T) = y \). \( V : \Omega \rightarrow \mathbb{R}^d \) is the geodesic flow and is equal to the gradient of the distance w.r.t the metric. In Riemannian case we have:

\[
V(y) = M(y)^{-1} \nabla u(y) \text{ where } \nabla \text{ is the normal Euclidean gradient.}
\]

This is based on the observation that the geodesics should always follow the direction of the greatest slope of the value function w.r.t the metric.

We extract the exact shortest path of every node in \( Y_i \) after \( y_l \) by (21). Then we search for the passive point till the finish of phase 1 as described in 4.2.

At the beginning of Phase 2, we get a set of passive nodes \( Y_i^* \). Every \( y_j^* \in Y_i^* \) the geodesic from \( x_i \) to \( y_j^* \) has been calculated. In stead of adding the highway edges into the next level,
we intend to add the discretized grid where the highway edges lied in to the next level (because it contains enough nodes and edges for the exact paths): We check the intersection of \( N(x_i) \) and \( N(y^*_j) \), by the property of passive, there should be at most one grid in the intersection. And thus we collect those grids as the next level’s discretization grid.

Another way to overcome this difficulty without the computation of geodesic is to consider this linear interpolation approximation as a stochastic optimal control problem. For example if we are in node \( x \), with it’s polytope neighborhood \( V(x) \).

4.4 The Finite Differences Highway Hierarchies

In this section we describe the Highwat Hierarchies based on the finite differences fast-marching method. Contrary to the Semi-Lagrangian scheme, this numerical scheme directly discretize the eikonal PDE using an upwind finite differences. We first recall the finite-difference fast-marching method:

For every \( x \in \Omega \), and every vector \( w \in \mathbb{E} \) and co-vector \( v \in \mathbb{E}^*_d \), if we define the Lagrangian and Hamiltonian as following:

\[
\mathcal{L}_x(w) := \frac{1}{2} \mathcal{F}_x(w)^2, \quad \mathcal{H}_x(v) := \sup_{w \in \mathbb{E}} \langle v, w \rangle - \mathcal{L}_x(w).
\]

Then the eikonal equation (18) can be written as the forms of Hamiltonian:

\[
2\mathcal{H}_x(\nabla u(x)) = 1, \forall x \in \Omega.
\]

We can check that the Hamiltonian is the half squared dual metric: \( \mathcal{H}_x = \frac{1}{2}(\mathcal{F}_x^*)^2 \), and thus the equivalent expression for \( \forall x \in \Omega, w \in \mathbb{E}, v \in \mathbb{E}^*_d \), \( \mathcal{H}_x(v) \) has the following form:

\[
\mathcal{H}_x(v) := \frac{1}{2} \sup_{w \in \mathbb{E}} \{ \langle v, w \rangle^2 \}.
\]

A good approximation of (21) is

\[
2\mathcal{H}_x(v) \approx 2\mathcal{H}_x(v) = \max_{i \in I} \sum_{j \in J} \rho_{i,j}(x) \langle v, \alpha_{i,j}(x) \rangle^2
\]

Where the choice of weights \( \rho_{i,j}(x) \) and the offsets \( \alpha_{i,j}(x) \) are depended one the form of metric. And it should be directly discretized in the tencil of our the numerical scheme, i.e.
∀x ∈ X, x − he_{i,j} ∈ X and x + he_{i,j} ∈ X. For simplify we ignore the dependency of the offsets and let e_{i,j} = α_{i,j}(x), and h is the discretization grid scale. Based on these, we can directly discrete the eikonal equation as following:

\[ \max_{i \in I} \sum_{j \in J} \rho_{i,j}(x) \{ U(x) - U(x - he_{i,j}) \}^2 = h^2, \quad \forall x \in X. \quad (27) \]

With boundary conditions U(x_0) = 0, for every x_0 ∈ ∂X. We can see that the left hand side of (26) is a non-decreasing function, and thus the discretization method is monotone and causal. Then it can be solved by the fast-marching method by the fixed-point operator.

In most of the cases we deal with the symmetric terms of (25), and thus we introduce a symmetric upwind finite differences of (26):

\[ \mu_{i,j}(x) \max \{ 0, U(x) - U(x - he_{i,j}), U(x) - U(x + he_{i,j}) \}^2, \quad \mu_{i,j}(x) \langle v, \alpha_{i,j}(x) \rangle^2. \quad (28) \]

And the upwind discretization scheme is then based on the stencil which is the convex hull of \{ x ± he_{i,j} \}_{j=1}^d. We denote the fixed point operator for finite-differences scheme by ∨_{FD}. Then the construction of the highway network is strictly an iterative scheme as follow:

- In the current level \( l \), for each node \( x_i \), we first do a phase 1 search by input the operator ∨_{FD} for the eikonal equation, then we will get a set of points \( Y_i = \{ y_1, y_2, ... \} \) that the search has passed through. The shortest path from the source point \( x_i \) to every point \( y_i \in Y_i \) is determined due to our direct discretization of the eikonal equation. As every iteration we calculate the value function we add a discretized edge into the shortest path tree.

- After we go out of the highway neighborhood \( N^l(x) \) of \( x_i \), we then start to do the abort condition test till definition 5 is satisfied, in order to get a set of passive nodes \( Y_i^* \). Then we do the phase 2 search to select the edges added to the next level. We do it iteratively from level to level.

In the following we give a sketch of the construction of highway structure:

**Algorithm 3 Highway Network for Fast Marching**

**Input:** Original grid, operator ∨, parameter H

**Output:** Highway Hierarchies \( X_0 \cup \partial X_0, ..., X_L \cup \partial X_L \)

**Initialization:** The original discretization grid.

1. while \( l < L \) do
2. \( E_{l+1} = \emptyset; \)
3. for \( \forall x_i \in V_l \) do
4. \( \text{set } u \leftarrow \land(+\infty), \text{ tag all points as non-accepted.} \)
5. \( \text{tag } x_i \text{ is active;} \)
6. while \( \text{There is non-accepted points} \) do
7. \( \text{Let } x \in X \cup \partial X \text{ be a minimizer of } u \text{ among non-accepted points;} \)
8. \( \text{tag } x \text{ as active;} \)
9. for \( \forall y \) non-accepted and such that and s.t. \( \land u(y) \) depends on \( x \) do
10. \( u(y) \leftarrow \land u(y); \)

20
The Query Algorithm:

- Suppose we want to find the shortest from \( s \) to \( t \), we do the normal finite-difference fast marching from \( x \) in the original \((X^0 \cup \partial X^0)\) as long as we stay in \( H^0(x) \). When we cross the highway neighborhood by edge \((x^*, y^*)\), we check the level of the node \( l(y^*) \). If \( l(y^*) > 0 \), then we change the search space from \((X^0 \cup \partial X^0)\) to \((X^1 \cup \partial X^1)\).

- We then start the normal finite-difference fast marching started from \( y^* \) in the level-1 discretization space \((X^1 \cup \partial X^1)\) till next time cross the highway neighborhood. We do this iteratively from level to level.

Notice that we admit a bidirectional query algorithm, i.e. the searches started from both the source point and the target point. When the searches meet in certain level then the minimal path is determined.

An effective accelerated method for the query is to do a more pre-process phase: After the construction of highway structure, we then find all the minimal paths from every node to every node in the highest level of the highway hierarchy \((X^L \cup \partial X^L)\). Then as long as both the searches reach the highest level, we can directly give the remained paths. But normally if we want to find the minimal paths from a certain set of nodes to all other nodes, it’s more efficiently to do the following:

**Continuous of Query Algorithm.**

When a highway fast-marching search reaches the highest level of the highway structure \((X^L \cup \partial X^L)\), denote the reached node \( x_L \). We then do a normal fast-marching started from
\( x_L \) to find the minimal paths to all other nodes in \((X^L \cup \partial X^L)\). After when the backward search reaches \((X^L \cup \partial X^L)\), we can directly give the remained paths. That is also what we adapt in the following applications.

4.5 Summary

First notice that our current working of the *Highway hierarchies* properties for the eikonal equations are heavily based on the discretization space, thus the analysis of the discretization methods counts a lot for the performance of the algorithm. Currently we are trying to make the problem more similar to the discrete shortest path problem by approximation and direct discretization of the Hamiltonian, in which case we can directly use the method to select the highway edges. But the finite-differences or semi-lagrangian discretized space of the eikonal equation are far different from the real world’s road network. These difference causes the main difficulties for the implementation of the algorithm. In fact in the later period of the internship we are trying to find a more suitable discretization methods to better match the idea of “highways”, and this will continue as a future work.

Based on the current discretization method, what we can do is to find the cases that could match the pre-processes phases and the selection method of *highway edges*. So in the original problem, there should be some inherent *highways*, and it should not cost a lot to find those highways. This leads to our following applications.
5 Application and Numerical Experiments

In this section, we give some results of the numerical tests in application. The algorithm is implemented in MATLAB, it’s a simple demo and we will advance it in the future work.

5.1 Test case 1

We consider a minimal time optimal control problem in dimension 2, where \( \Omega \subseteq \mathbb{R}^2 \) with the control \( \|\alpha\|_2 \leq 1 \) and for every \( x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \in \Omega \), the dynamics are as following:

\[
\begin{aligned}
\dot{x}_1(t) &= \alpha_1(t) c \\
\dot{x}_2(t) &= \alpha_2(t) e^{bx_1}
\end{aligned}
\]  

(29)

where \( c, b \) are constants. Denoted by \( x(t; \alpha_t, x_0) \) the solution of dynamics (29) corresponding to the control \( \alpha(t) \) and the initial condition \( x_0 \) and let \( c(x) = \begin{pmatrix} c \\ be^{x_1} \end{pmatrix} \). Then the minimum time function \( T(x) \) corresponding to the viscosity solution of the eikonal equation:

\[
\begin{aligned}
c_1(x)^2 \left( \frac{\partial T}{\partial x_1} \right)^2 + c_2(x)^2 \left( \frac{\partial T}{\partial x_2} \right)^2 &= 1, \quad x \in \Omega \setminus \Omega_0 \\
T(x) &= 0, \quad x \in \Omega_0
\end{aligned}
\]  

(30)

The physical interpretation of this eikonal equation is in the front propagation problem in 2-dimensions, in which the interface propagates in direction \( x_1 \) with a constant speed \( c \), and propagates in direction \( x_2 \) with the speed \( be^{x_1} \) which increases exponential as the \( x_1 \) increase.

There should be inherent some “highways” in the propagations. For instance, the current position is \( (x_1, x_2) \) and we want to find the minimal time to \( (x_1', x_2') \). If those positions are close, a trivial solution is straightly moving from \( x_2 \) to \( x_2' \) with \( \dot{x}_1 = \alpha(t)c \) without the change of \( x_1 \). However when those positions are far away i.e. \( x_2' \gg x_2 \), there is another way may decrease the time: first move from \( (x_1, x_2) \) to \( (x_1', x_2) \) with \( \dot{x}_1 = \alpha c \), than move from \( (x_1, x_2) \) to \( (x_1, x_2') \) with \( \dot{x}_2 = \alpha e^{bx_1} \), than move back from \( (x_1', x_2') \) to \( (x_1, x_2) \) with \( \dot{x}_1 = \alpha c \). It means that the speed of direction \( x_2 \) in \( x_1' \) is greater than in \( x_1 \) such that the propagating time spend in direction \( x_2 \) can make up the lost time moving in direction \( x_1 \). In that case \( (x_1', x_2) \) to \( (x_1, x_2') \) is a ”highway” compare to \( (x_1, x_2) \) to \( (x_1', x_2') \).

Our highway hierarchies algorithm aims to find those highways, and collect them as a structure such that, when the propagation reaches a certain level, we will focus the computation on the search in these highways.
5.2 Numerical results for test case 1

In the numerical experiments, we fix $b = 3$ in the equation (25). The highway neighborhood parameters $H$ takes 15 in every element $H_t \in H$, and we construct a 5 levels highway structures. Then we change $c$ to see how our algorithm works for different situation of the speed in $x_1$. The discretization size take $200 \times 200$.

We are doing the numerical experiments in MATLAB, with a Intel core i7-9750H CPU@2.60GHz and 16GB RAM.

<table>
<thead>
<tr>
<th>number of $c$</th>
<th>1</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges of level-1</td>
<td>78670</td>
<td>78120</td>
<td>76220</td>
<td>68130</td>
<td>42780</td>
</tr>
<tr>
<td>edges of level-2</td>
<td>76780</td>
<td>76167</td>
<td>72150</td>
<td>59280</td>
<td>39320</td>
</tr>
<tr>
<td>edges of level-3</td>
<td>74740</td>
<td>73880</td>
<td>68760</td>
<td>52670</td>
<td>35670</td>
</tr>
<tr>
<td>edges of level-4</td>
<td>72620</td>
<td>71160</td>
<td>65130</td>
<td>47180</td>
<td>31780</td>
</tr>
<tr>
<td>edges of level-5</td>
<td>70320</td>
<td>67231</td>
<td>60180</td>
<td>43250</td>
<td>28160</td>
</tr>
<tr>
<td>Pre-process(CPU Time)</td>
<td>1369.5</td>
<td>1301.8</td>
<td>1261.3</td>
<td>1189.8</td>
<td>897.7</td>
</tr>
<tr>
<td>Fast-marching(CPU Time)</td>
<td>4.538</td>
<td>4.534</td>
<td>4.541</td>
<td>4.533</td>
<td>4.539</td>
</tr>
<tr>
<td>Highway Hierarchy FM(CPU Time)</td>
<td>4.517</td>
<td>4.498</td>
<td>4.121</td>
<td>3.671</td>
<td>2.471</td>
</tr>
</tbody>
</table>

Analysis of the result: As we can see in the sheet, when $c$ is small our algorithm almost has no improvement, that is because the gap between the speed in $x_1$ and $x_2$ are not big. In this case to find the highway will cost too much and even the query proprocess will search almost the same space. But when $c$ is increasing, i.e. the speed in $x_1$ is enough fast such that it is worth to find the ”highway”, since the time spend in $x_1$ is negligible compare to in $x_2$.

In fact, if we take $c$ enough big, see the test case $c = 100$, then the highway will be all the paths in $x_1$ direction and the only path in $x_2$ with the biggest value of $x_1$. Thus after the selection of these highways, in most of the iterative step of the fast-marching search based on the highway structure will only calculate the value in the highway, and thus reduce almost $\frac{2}{3}$ of calculation. see the figure as following:

As show in the figure, normal when the fast-marching propagate at $(x_i^1, x_j^{2-1})$, we need to calculate all values of it's neighborhood nodes $(x_i^1, x_j^2)$, $(x_i^1, x_{j+1}^2)$. However after the pre-process of our highway hierarchy, we know that when we reach the level of $(x_i^1, x_j^2)$ highway structure, the only possible direction to propagate is to $(x_i^{1+1}, x_j^2)$, and thus we reduce the needed calculations.

Also we attached the result of then distance map with the search started in the upper left,
with $c = 1$ (up-left), $c = 10$ (up-right) and $c = 100$. 
6 Conclusion

6.1 The Summary of Current Work

During the internship we first studied several shortest path algorithm in discrete cases, and especially the Highway Hierarchies method for the shortest path problem. We then try to introduce this kind of method to continuous time and state problem. We focus on the Eikonal Equations based on the observation that it has a lot in common with the shortest path problem. We then studied several numerical solutions for the eikonal equation, mainly on the fast-marching method, from the isotropical cases to more general anisotropical cases.

After we try to implement the highway hierarchies method to the fast-marching algorithm for the eikonal equation, based on two discretezation method: the semi-lagrangian fast-marching and the finite-difference fast-marching. We introduce a new highway-hierarchy fast-marching algorithm, and then experiment numerically in a particular application case.

6.2 Proposed Future Work

While during the last period of the internship, we found that it’s difficult to directly apply the highway hierarchy to the discretization grid. This is because there are many differences between the real world’s road network and the discretization space of the eikonal PDEs. Especially for the efficiency of our algorithm, it needs the discretization space has several properties, which seems really hard to be satisfied. In the further work we would try more other discretization methods to overcome this kind of difficulties, and may combine with other numerical methods.

Our current works are more on the algorithms aspects, in further work we would like to work more from the mathematical point of view. In fact the idea of the ”highway structure” is really interesting, and we would like to further discover the ”highway structure” of the PDEs and control problems themself, not only based on the discretization.

We could like to adapt the Highway Hierarchies and work on more general continuous time and state optimal control problems, not only the eikonal equations.

6.3 Acknowledgement

First of all, I would like to thank Professor Marianne AKIAN and Professor Stephane GAUBERT, who give me a great opportunity to start my research work in this interesting topic, and I am also grateful to be fortunate enough continuing to do research under their guidance.

I especially thank Professor Marianne AKIN, who managed time almost every week to meet me in visio or in face during these special and hard period time of lockdown, and who also gave me all-round guidance and inspiration during the internship.

I would like to thank the organizers and the professors in master-optimisation, which provided us a good training in mathematical optimization and related topics, and also gave us various opportunities for seminar and conference.
References


