

Causal Domain Restriction for Eikonal Equations

Z. Clawson^{†,††}, A. Chacon^{‡,††}, A. Vladimirovsky^{††}

Center for Applied Mathematics and Department of Mathematics
Cornell University, Ithaca, NY 14853

Abstract

Many applications require efficient methods for solving continuous shortest path problems. Such paths can be viewed as characteristics of static Hamilton-Jacobi equations. Several fast numerical algorithms have been developed to solve such equations on the whole domain. In this paper we consider a somewhat different problem, where the solution is needed at one specific point, so we restrict the computations to a neighborhood of the characteristic. We explain how heuristic under/over-estimate functions can be used to obtain a *causal* domain restriction, significantly decreasing the computational work without sacrificing convergence under mesh refinement. The discussed techniques are inspired by an alternative version of the classical A* algorithm on graphs. We illustrate the advantages of our approach on continuous isotropic examples in 2D and 3D. We compare its efficiency and accuracy to previous domain restriction techniques. We also analyze the behavior of errors under the grid refinement and show how Lagrangian (Pontryagin's Maximum Principle-based) computations can be used to enhance our method.

1 INTRODUCTION

The Eikonal equation

$$\begin{cases} |\nabla u(\mathbf{x})| f(\mathbf{x}) &= 1 & \forall \mathbf{x} \in \Omega \\ u(\mathbf{x}) &= q(\mathbf{x}) & \forall \mathbf{x} \in \mathcal{Q} \subseteq \partial\Omega, \end{cases} \quad (1.1)$$

arises naturally in many applications including continuous optimal path planning, computational geometry, photolithography, optics, shape from shading, and image processing [24]. One natural interpretation for the solution of (1.1) comes from isotropic time-optimal control problems. For a vehicle traveling through Ω , f describes the speed of travel and q gives the exit time-penalty charged on \mathcal{Q} . In this framework, $u(\mathbf{x})$ is the *value function*; i.e., the minimum time to exit Ω through \mathcal{Q} if we start from a point $\mathbf{x} \in \Omega$. The characteristic curves of the PDE (1.1) define the optimal trajectories for the vehicle motion.

The value function is Lipschitz continuous, but generally is not smooth on Ω . (The gradient of u is undefined at all points for which an optimal trajectory is not unique.) Correspondingly, the PDE (1.1) typically does not have a smooth solution and admits infinitely many Lipschitz continuous weak solutions. Additional conditions introduced in [9] are used to restore the uniqueness: the *viscosity solution* is unique and coincides with the value function of the above control problem.

In the last 20 years, many fast numerical methods have been developed to solve (1.1) *on the entire domain* $\bar{\Omega}$; e.g., see [8, 23, 27, 31]. Many of these fast methods were inspired by classical label-correcting and label-setting algorithms on graphs; e.g., Sethian's Fast Marching Method [23] mirrors the logic of the classical Dijkstra's algorithm [10], which finds the minimum time to a target-node from every other node in the graph.

Our focus here is on a somewhat different situation, with the solution needed *for one specific starting position only*. On graphs, an A* modification of Dijkstra's method [13] is widely used for similar *single source / single target* shortest path problems. There have been several prior attempts to extend A* techniques to algorithms for continuous optimal trajectory problems, but all of them

[†]SUPPORTED IN PART BY THE NSF THROUGH GRADUATE RESEARCH FELLOWSHIP AND THE 2010 REU AT CORNELL.

[‡]SUPPORTED IN PART BY ALFRED P. SLOAN FOUNDATION GRADUATE FELLOWSHIP.

^{††}SUPPORTED IN PART BY THE NATIONAL SCIENCE FOUNDATION GRANT DMS-1016150.

have significant drawbacks: these methods either produce additional errors that do not vanish under numerical grid refinement [15, 16, 17, 18], or provide much more limited computational savings [11, 29, 30]. We believe that these disadvantages stem from an overly faithful mirroring of the “standard” A* on graphs. Our own approach is based on an alternative version of the A* algorithm [5] that has clear advantages in continuous optimal control problems. Numerical testing confirms that our method is both efficient (in terms of the percentage of domain restriction) and convergent under grid refinement.

We begin by reviewing two flavors of A* techniques on graphs in §2. We then describe the standard Fast Marching Method and its various A*-type modifications in §3. The numerical tests in §4 are used to compare the efficiency and accuracy of competing domain restriction techniques. We discuss the limitations of our approach and directions of future work in §5. The Appendix (§6) contains convergence analysis of the alternative A* under grid refinement, exploiting the probabilistic interpretation of the discretized equations.

2 DOMAIN RESTRICTION TECHNIQUES ON GRAPHS

We start by defining the shortest path problem on a graph:

- A graph \mathcal{G} is defined by a set of nodes (vertices) $X = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{M+1} = \mathbf{t}\}$ and a set of directed arcs between these nodes.
- Along each arc we prescribe a transition time penalty $C(\mathbf{x}_i, \mathbf{x}_j) = C_{ij} > 0$, and assume $C_{ij} = +\infty$ if there is no transition from \mathbf{x}_i to \mathbf{x}_j .
- The sets of *in-neighbors* and *out-neighbors* of a node \mathbf{x}_j are respectively defined by

$$N_j^- = N^-(\mathbf{x}_j) \triangleq \{\mathbf{x}_i \mid C_{ij} < +\infty\}, \quad N_j^+ = N^+(\mathbf{x}_j) \triangleq \{\mathbf{x}_k \mid C_{jk} < +\infty\}.$$

- Assume the graph is *sparsely connected*, i.e. $|N^\pm(\mathbf{x}_i)| \leq \kappa \ll M \ \forall \ \mathbf{x}_i \in X$ for some fixed $\kappa \in \mathbb{N}$.
- The **goal** is to find the “*value function*” $U : X \rightarrow [0, +\infty)$, defined as

$$U_i = U(\mathbf{x}_i) \triangleq \text{the minimum total time to travel from } \mathbf{x}_i \text{ to } \mathbf{t} = \mathbf{x}_{M+1}.$$

Naturally, $U_{M+1} = 0$. On the rest of the graph, *Bellman’s Optimality Principle* [4] yields a coupled system of M nonlinear equations:

$$U_i = \min_{\mathbf{x}_j \in N_i^+} \{U_j + C_{ij}\}, \quad \forall \ i = 1, 2, \dots, M. \quad (2.1)$$

Once the value function is known, an optimal path from any node to the target \mathbf{x}_{M+1} can be quickly recovered by recursively transitioning to the minimizing neighbor. A straight-forward iterative method for solving the system (2.1) would result in $O(M^2)$ computational cost. Fortunately, this system is *monotone causal*: U_i cannot depend on U_j unless $U_i > U_j$. This observation is the basis of the classical Dijkstra’s method, which recovers the value function on the entire graph in $O(M \log M)$ operations [10]. In Dijkstra’s method, all nodes are split into three classes: **FAR** (no value yet assigned), **CONSIDERED** (assigned a tentative value), or **ACCEPTED** (assigned a permanent value).

Algorithm 1: *Dijkstra's Algorithm*

Initialization:

- 1 $U_i \leftarrow +\infty$ and mark \mathbf{x}_i as **FAR** for $i = 1, 2, \dots, M$
- 2 $U(\mathbf{t}) \leftarrow 0$ and mark \mathbf{t} as **ACCEPTED**.
- 3 For all $\mathbf{x}_i \in N^-(\mathbf{t})$, mark as **CONSIDERED** and $U_i \leftarrow C(\mathbf{x}_i, \mathbf{t})$

Algorithm :

- 4 **while** \exists a **CONSIDERED** node **do**
 - 5 Find the **CONSIDERED** node \mathbf{x}_j with minimal U -value and mark as **ACCEPTED**
 - 6 **for** $\mathbf{x}_i \in N_j^-$ such that $U_i > U_j$ and \mathbf{x}_i is **FAR** or **CONSIDERED** **do**
 - 7 $\tilde{U} \leftarrow U_j + C_{ij}$
 - 8 **if** $\tilde{U} < U_i$ **then**
 - 9 $U_i \leftarrow \tilde{U}$
 - 10 Mark \mathbf{x}_i as **CONSIDERED**
-

Efficient implementations usually maintain the **CONSIDERED** nodes as a binary heap, resulting in the $\log M$ term in the computational complexity.

2.1 ESTIMATES FOR “SINGLE-SOURCE / SINGLE-TARGET” PROBLEMS.

If we are only interested in an optimal path from a single starting location $\mathbf{s} \in X$, Dijkstra's method can be terminated as soon \mathbf{s} becomes **ACCEPTED**. (This changes the stopping criterion on line 4 of the pseudocode.) Other modifications of the algorithm can be introduced to further reduce the computational cost on this narrower problem. Consider a function

$$V_i = V(\mathbf{x}_i) \triangleq \text{minimum total time to travel from } \mathbf{s} \text{ to } \mathbf{x}_i.$$

Any node \mathbf{x}_i lying on an optimal path from \mathbf{s} to \mathbf{t} must satisfy $U_i + V_i = U(\mathbf{s}) = V(\mathbf{t})$. This provides an obvious relevance criterion, since for any \mathbf{x}_i that is not on an optimal path, $U_i + V_i > U(\mathbf{s})$. But since V is generally unknown, all techniques for focusing computations on a neighborhood of this optimal path must instead rely on some “heuristic underestimate”

$$\varphi_i = \varphi(\mathbf{x}_i) \leq V_i. \quad (2.2)$$

A stronger “consistency” requirement is often imposed instead:

$$\varphi_j \leq C_{ij} + \varphi_i; \quad \forall i, j. \quad (2.3)$$

(Note that $\varphi \equiv V$ is the maximum among all consistent heuristics that also satisfy $\varphi(\mathbf{s}) = 0$.)

Such consistent underestimates are readily available for geometrically embedded graphs. Suppose $X \subset \mathbb{R}^n$ and $d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|_2$. If the “maximum speed” $F_2 > 0$ is such that $C_{ij} \geq d_{ij}/F_2$ for all i and j , then $\varphi_i = \|\mathbf{x}_i - \mathbf{s}\|_2/F_2 \leq V_i$. On a Cartesian grid-type graph, the Manhattan distance provides a better (tighter) underestimate $\varphi_i = \|\mathbf{x}_i - \mathbf{s}\|_1/F_2$. For more general embedded graphs, a much better underestimate φ can be produced by “*Landmark sampling*” [12], but this requires additional precomputation and increases the memory footprint of the algorithm.

Some algorithms for this problem also rely on “heuristic overestimates”

$$\psi_i = \psi(\mathbf{x}_i) \geq V_i.$$

An overestimate can be obtained as a total cost of *any* path from \mathbf{x}_i to \mathbf{s} or can also be found using landmark precomputations [12]. For structured geometrically embedded graphs, an analytic expression might also be available. E.g., on a Cartesian grid, if the “minimum speed” $F_1 > 0$ is such that $C_{ij} \leq d_{ij}/F_1$, we can use $\psi_i = \|\mathbf{x}_i - \mathbf{s}\|_1/F_1$.

2.2 A* RESTRICTION TECHNIQUES

A* techniques restrict computations to potentially relevant nodes by limiting the number of nodes that become **CONSIDERED**. A more accurate φ restricts a larger number of nodes from becoming **CONSIDERED**, and if $\varphi = V$ then only those nodes actually on the $\mathbf{s} \rightarrow \mathbf{t}$ optimal path are ever **ACCEPTED**.

Standard A* (SA*). This version of A* is the one most often described in the literature [13]. Unlike in Dijkstra’s algorithm, the **CONSIDERED** nodes are sorted and **ACCEPTED** based on $(U_i + \varphi_i)$ values. This change affects line 5 in our pseudocode. The resulting algorithm typically **ACCEPTS** far fewer nodes before terminating: irrelevant nodes with large φ values might still become **CONSIDERED** (if their neighbors are accepted) but will have lower priority and most of them will never become **ACCEPTED** themselves. Moreover, the consistency of φ ensures that **ACCEPTED** nodes receive exactly the same values as would have been produced by the original Dijkstra’s method. If \mathbf{x}_i actually depends on $\mathbf{x}_j \in N_i^+$, then

$$U_i = C_{ij} + U_j \implies U_i \geq (\varphi_j - \varphi_i) + U_j \iff U_i + \varphi_i \geq U_j + \varphi_j,$$

guaranteeing that under SA* \mathbf{x}_i will not be **ACCEPTED** before \mathbf{x}_j .

Alternative A* (AA*). A less common variant of A* is described in [5]. Instead of favoring nodes with small φ , AA* simply ignores nodes that are clearly irrelevant. AA* relies on an underestimate φ (no longer required to satisfy (2.3)) and an additional upper bound $\Psi \geq U(\mathbf{s})$. (If an analytic or precomputed ψ is available, we can take $\Psi = \psi(\mathbf{t})$. But it is also possible to use the total cost of any feasible path from \mathbf{s} to \mathbf{t} .)

During Dijkstra’s algorithm, a node \mathbf{x}_i with $U_i + \varphi_i > \Psi$ (hence $U_i + V_i > \Psi$) is surely not a part of the optimal path. Thus, to speed up Dijkstra’s algorithm, in AA* we still sort **CONSIDERED** nodes based on U values, but on line 10 we only mark \mathbf{x}_i **CONSIDERED** if $U_i + \varphi_i \leq \Psi$. Since the order of acceptance is the same, it is clear that AA* produces the same values as Dijkstra’s, but the efficiency of this technique is clearly influenced by the quality of Ψ (the smaller it is, the smaller is the number of **CONSIDERED** nodes). This reliance on Ψ is a downside (since SA* only needs φ), but has the advantage of making AA* also applicable to the label-correcting methods [5]. In section §3 we argue that AA* is also more suitable for continuous optimal control problems, in which an Ψ is often readily available.

AA* with Branch & Bound (B&B). In AA* Ψ remains static throughout the algorithm. The idea of *Branch & Bound* (B&B) is to dynamically decrease Ψ as we gain more information about the graph, making use of an overestimate function ψ . When **ACCEPTING** a node \mathbf{x}_i , we can also set

$$\Psi \leftarrow \min \{ \Psi, U_i + \psi_i \}.$$

Exact estimates. Using “exact estimates” with A* would result in the maximal domain restriction. For both A* techniques, if $\varphi \equiv V$ and $\Psi \equiv U(\mathbf{s})$, the algorithm would only **ACCEPT** the nodes lying on an optimal path.

3 DOMAIN RESTRICTION IN A CONTINUOUS SETTING

The continuous time-optimal isotropic control problem deals with minimizing the time-to-exit for a vehicle, whose dynamics is governed by

$$\begin{cases} \dot{\mathbf{y}}(t) &= f(\mathbf{y}(t)) \mathbf{a}(t), \\ \mathbf{y}(0) &= \mathbf{x} \in \Omega \subset \mathbb{R}^n, \end{cases} \quad (3.1)$$

Here \mathbf{x} is the starting position, $\mathbf{a}(t) \in S^{n-1}$ is the control (i.e., the direction of motion) chosen at the time t , $\mathbf{y}(t)$ is the vehicle’s time-dependent position, and f is the spatially-dependent speed of motion. We will further assume the existence of two constants F_1 and F_2 such that $0 < F_1 \leq f(\mathbf{x}) \leq F_2$

holds $\forall \mathbf{x} \in \bar{\Omega}$. For every time-dependent control $\mathbf{a}(\cdot)$ we define the total time to the exit set $\mathcal{Q} \subseteq \partial\Omega$ as $T_{\mathbf{x},\mathbf{a}} = \min \{t \geq 0 \mid \mathbf{y}(t) \in \mathcal{Q}\}$. The *value function* $u : \Omega \rightarrow [0, +\infty)$ is then naturally defined as

$$u(\mathbf{x}) = \inf_{\mathbf{a}(\cdot)} \{T_{\mathbf{x},\mathbf{a}} + q[\mathbf{y}(T_{\mathbf{x},\mathbf{a}})]\},$$

where $q : \mathcal{Q} \rightarrow [0, +\infty)$ is the exit-time penalty. Bellman's optimality principle can be used to show that, if u is a smooth function, it must satisfy a static Hamilton-Jacobi-Bellman PDE

$$\min_{\mathbf{a} \in A} \{(\nabla u(\mathbf{x}) \cdot \mathbf{a}) f(\mathbf{x}) + 1\} = 0,$$

with the natural boundary condition $u = q$ on \mathcal{Q} . Using the isotropic nature of the dynamics, it is clear that the minimizer (i.e., the optimal initial direction of motion starting from \mathbf{x}) is $\mathbf{a}_* = -\nabla u(\mathbf{x}) / \|\nabla u(\mathbf{x})\|$ and the equation is equivalent to the Eikonal PDE (1.1). If the value function u is not smooth, it can still be interpreted as a unique *viscosity solution* of this PDE [9].

Solving this PDE to recover the value function is the key idea of the *dynamic programming*. An analytic solution is usually unavailable, so numerical methods are needed to approximate u . We use a first-order upwind discretization, whose monotonicity and consistency yield convergence to the viscosity solution [3]. To simplify the notation, we describe everything on a cartesian grid in \mathbb{R}^2 , though higher dimensional generalizations are straightforward and similar discretizations are also available on simplicial meshes (e.g., [14, 25]; see also Figure 3). We will assume

- $\bar{\Omega} = [0, 1] \times [0, 1]$ is discretized on a $m \times m$ uniform grid X with spacing $h = 1/(m-1)$.
- A *gridpoint* or *node* is denoted by \mathbf{x}_{ij} with corresponding *value* $U_{ij} = U(\mathbf{x}_{ij}) \approx u(\mathbf{x}_{ij})$, *speed* $f_{ij} = f(\mathbf{x}_{ij})$, and *neighbors*

$$N_{ij} = N(\mathbf{x}_{ij}) \triangleq \{\mathbf{x}_{i-1,j}, \mathbf{x}_{i+1,j}, \mathbf{x}_{i,j-1}, \mathbf{x}_{i,j+1}\}.$$

This notation will be slightly abused (e.g., a gridpoint \mathbf{x}_i with a corresponding value U_i , speed f_i , etc...) whenever we emphasize the ordering of gridpoints rather than their geometric position.

- We will assume that $\mathcal{Q} \subseteq \partial\Omega$ is *well-discretized* on the grid, and we define the *discretized exit-set* $Q = \mathcal{Q} \cap X$. In particular, the focus of our computational experiments will be on the case $Q = \{\mathbf{t}\}$ with the exit time-penalty $q(\mathbf{t}) = 0$. We note that \mathbf{t} does not have to be on the boundary of the square $\bar{\Omega}$: if $\mathbf{t} \in (0, 1)^2$, then $\Omega = (0, 1)^2 \setminus \{\mathbf{t}\}$, and the border of the square is treated as an essentially outflow boundary. This corresponds to solving a $\bar{\Omega}$ -constrained optimal control problem, with u interpreted as a constrained viscosity solution [2].

We use the upwind finite differences [22] to approximate the derivatives of (1.1), resulting in a system of discretized equations. Using the standard four-point nearest-neighbors stencil at each $\mathbf{x}_{ij} \in X$, this results in:

$$(\max \{D^{-x}U_{ij}, -D^{+x}U_{ij}, 0\})^2 + (\max \{D^{-y}U_{ij}, -D^{+y}U_{ij}, 0\})^2 = \frac{1}{f_{ij}^2}, \quad (3.2)$$

$$\text{where } u_x(x_i, y_j) \approx D^{\pm x}U_{ij} = \frac{U_{i\pm 1,j} - U_{ij}}{\pm h}, \quad \text{and } u_y(x_i, y_j) \approx D^{\pm y}U_{ij} = \frac{U_{i,j\pm 1} - U_{ij}}{\pm h}.$$

If all the neighboring values are known, this is really a “quadratic equation in disguise” for U_{ij} . Letting $U_H = \min \{U_{i-1,j}, U_{i+1,j}\}$ and $U_V = \min \{U_{i,j-1}, U_{i,j+1}\}$ reduces (3.2) to

$$(U_{ij} - U_H)^2 + (U_{ij} - U_V)^2 = \frac{h^2}{f_{ij}^2}, \quad (3.3)$$

provided the solution satisfies $U_{ij} \geq \max \{U_H, U_V\}$; otherwise we perform a *one-sided update*:

$$U_{ij} = \min \{U_H, U_V\} + \frac{h}{f_{ij}}. \quad (3.4)$$

The system of discretized equations ((3.3) and (3.4) for all (i, j)) are *monotone causal* since U_{ij} needs only its *smaller* neighboring values to produce an update.

Sethian’s **Fast Marching Method (FMM)** [23] and another Dijkstra-like algorithm [27] due to Tsitsiklis take advantage of this monotone causality. FMM can be obtained from Dijkstra’s Method by changing the lines 3 and 7 to instead use the continuous update procedure (equations (3.3) and (3.4)). Similarly to Dijkstra’s method, FMM computes the value function on the entire grid in $O(M \log M)$ operations, where $M = m^2$ is the number of gridpoints. The key question is whether a significant reduction of computational cost is possible if we are only interested in an optimal trajectory starting from a single (pre-specified) source gridpoint \mathbf{s} .

Remark 3.1. Restricting FMM to a smaller (relevant) subset of Ω via A*-techniques is precisely the focus of this paper. But a legitimate related question is whether the dynamic programming approach is at all necessary when a single trajectory is all that we desire? In contrast to path planning on graphs, in the continuous control community, optimal trajectories for single source problems are typically recovered via **Pontryagin Maximum Principle (PMP)** [21]. This involves solving a two point boundary value problem for a state-costate system of ODEs, which in our context could be also derived as characteristic ODEs of the Eikonal PDE (1.1). One advantage of using PMP is that, unlike the dynamic programming, it does not suffer from the *curse of dimensionality*. In higher dimensions, solving a two-point boundary value problem is much more efficient than solving a PDE on the whole domain. Unfortunately, PMP is harder to apply if the speed function f is not smooth. Even more unpleasantly, depending on the initial guess used to solve the two-point boundary value problem, that method often converges to *locally* optimal trajectories. In contrast, the dynamic programming always yields a globally optimal trajectory, and our approach can be used to lower its computational cost in higher dimensions. In fact, we show that both techniques can be used together, with a prior use of PMP improving the efficiency of A*, and A* verifying the global optimality of a PMP-produced trajectory.

3.1 DOMAIN RESTRICTION WITHOUT HEURISTIC UNDERESTIMATES.

Our algorithmic goal is to restrict FMM to a dynamically defined subset of the grid using underestimates of the cost-to-go and the previously computed values. This is the essence of several A*-type techniques compared in sections 3.2-3.5. But to motivate the discussion, we start by considering several simpler domain restriction techniques that do not involve the run-time use of underestimates.

First, we note that FMM can be terminated immediately after the gridpoint \mathbf{s} is **ACCEPTED**. In practice, this is unlikely to yield significant computational savings unless the set

$$L = \{\mathbf{x} \in \bar{\Omega} \mid u(\mathbf{x}) \leq u(\mathbf{s})\}$$

is much smaller than the entire $\bar{\Omega}$. (E.g., see the bolded level set ∂L in Figure 2A.)

Second, it is possible to use a “bi-directional FMM” (similar to the bi-directional Dijkstra’s [20]) by expanding two **ACCEPTED** clouds from the source and the target and stopping the process when they meet. The first gridpoint accepted in both clouds is guaranteed to lie on an $O(h)$ -suboptimal trajectory from \mathbf{s} to \mathbf{t} . This approach is potentially much more efficient than the above. E.g., for a constant speed function $f = 1$, it cuts the n -dimensional volume of the **ACCEPTED** set by the factor of 2^{n-1} ; see Figure 1.

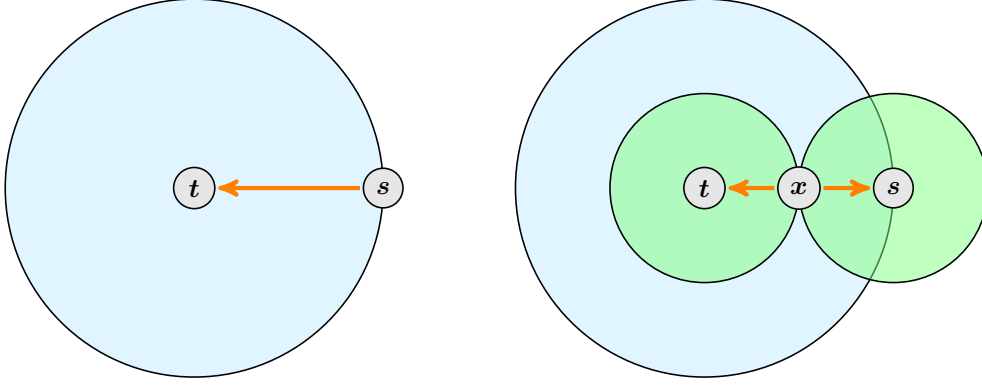


Figure 1: FMM expands computations outwards from \mathbf{t} , shown by the large circle. The two smaller circles each expand from \mathbf{t} and \mathbf{s} and represent the computations performed during BiFMM. In this simple situation BiFMM considers 50% of the domain that FMM considers. To recover the global optimal trajectory from \mathbf{s} to \mathbf{t} using BiFMM one must recover the optimal trajectories from \mathbf{x} to \mathbf{t} and \mathbf{x} to \mathbf{s} and join them together.

Third, a different “elliptical restriction” approach is also applicable (and can be combined with the above bidirectional technique) whenever an overestimate for the minimal time from \mathbf{s} to \mathbf{t} is available.

Lemma 3.2. *Suppose the exit-set is given by a single target point \mathbf{t} , $d = |\mathbf{s} - \mathbf{t}|$, and Ψ is a known constant such that $\Psi \geq u(\mathbf{s})$. Then the optimal trajectory $\mathbf{y}(\cdot)$ satisfying (3.1) from $\mathbf{y}(0) = \mathbf{s}$ is contained within the prolate spheroid $E(\mathbf{s}, \mathbf{t})$ (an ellipse in 2D) satisfying*

$$\text{Foci} = \{\mathbf{s}, \mathbf{t}\} \quad \text{and} \quad \begin{cases} \text{Major semi-axis} &= a &= \frac{F_2 \Psi}{2}, \\ \text{Minor semi-axis} &= b &= \frac{1}{2} \sqrt{F_2^2 \Psi^2 - d^2}. \end{cases} \quad (3.5)$$

Proof. Let d^* and T^* be the distance and time along the optimal trajectory from \mathbf{s} to \mathbf{t} . Then

$$\frac{d^*}{F_2} \leq T^* \leq \Psi, \quad (3.6)$$

For any \mathbf{x} along the optimal trajectory we have

$$|\mathbf{x} - \mathbf{s}| + |\mathbf{t} - \mathbf{x}| \leq d^* \leq F_2 \Psi.$$

This inequality defines a prolate spheroid in \mathbb{R}^n and (3.5) immediately follows. \square

Even if we are interested in an unconstrained problem (find the quickest (\mathbf{s}, \mathbf{t}) trajectory in \mathbb{R}^2), finite computer memory forces us to solve a *state-constrained* problem instead (find the quickest (\mathbf{s}, \mathbf{t}) trajectory contained in $\bar{\Omega}$). The above Lemma is thus also useful to answer a related question: for which starting points \mathbf{s} does the $\bar{\Omega}$ -constrained problem have the same value function as the unconstrained? Clearly, for any point \mathbf{s} such that $E(\mathbf{s}, \mathbf{t}) \subset \bar{\Omega}$, enlarging the domain would not decrease $u(\mathbf{s})$.

Higher dimensional savings. Restricting computations to $E(\mathbf{s}, \mathbf{t})$ has an increasing effect in higher dimensions. The fraction \mathcal{P} of the volume of $E(\mathbf{s}, \mathbf{t})$ to the volume of the smallest bounding rectangular box B is given by $\mathcal{P} = \frac{\pi^{n/2}}{2^n \Gamma(n/2 + 1)}$, which quickly approaches zero as n grows. (E.g., in \mathbb{R}^2 this fraction is $(\pi/4) \approx 78.5\%$, while in \mathbb{R}^6 it is already $\approx 8\%$.) If $\bar{\Omega} = B$, the restriction to $E(\mathbf{s}, \mathbf{t})$ yields the computational savings of $(1 - \mathcal{P})$; the savings are even higher if $\bar{\Omega}$ is any other box-rectangular domain fully containing $E(\mathbf{s}, \mathbf{t})$.

Formulas for Ψ can be naturally obtained by computing (or bounding from above) the time along any feasible path from \mathbf{s} to \mathbf{t} . On a convex domain Ω , the most obvious choice is $\Psi_1 = d/F_1$ (i.e.,

follow the straight line from \mathbf{s} to \mathbf{t} at the minimum speed F_1). For problems with the unit speed of motion, $f(\mathbf{x}) = 1 = F_1 = F_2$, $\Psi_1 = d$, and the ellipse collapses to a straight line segment.

A more accurate overestimate can be obtained by computing the exact time needed to traverse that straight line trajectory:

$$\Psi_2 = \int_0^{|t-s|} \frac{dr}{f\left(\mathbf{s} + \frac{t-s}{|t-s|}r\right)} = \int_0^1 \frac{|t-s|}{f(\mathbf{s} + (t-s)r)} dr \leq \Psi_1.$$

For non-convex domains, a similar upper bound can be obtained by integrating the slowness $1/f$ along any feasible trajectory (e.g., the shortest Ω -constrained path from \mathbf{s} to \mathbf{t}).

Finally, we will also consider the third (“ideal”) option, with $\Psi_3 = u(\mathbf{s}) \leq \Psi_2$. While practically unattainable, Ψ_3 is useful to illustrate the upper bound on efficiency of various domain restriction techniques. In practice, it can be approximated by using $U(\mathbf{s})$ precomputed on a coarser grid or using the output of Pontryagin-Maximum-Principle-based computations (see the example in section 4.4). In the latter case, the techniques discussed in this paper can be viewed as a method for verifying the global optimality of a known locally-optimal trajectory. Figure 2A shows the (\mathbf{s}, \mathbf{t}) -focused ellipses for a specific example with a highly oscillatory speed function.

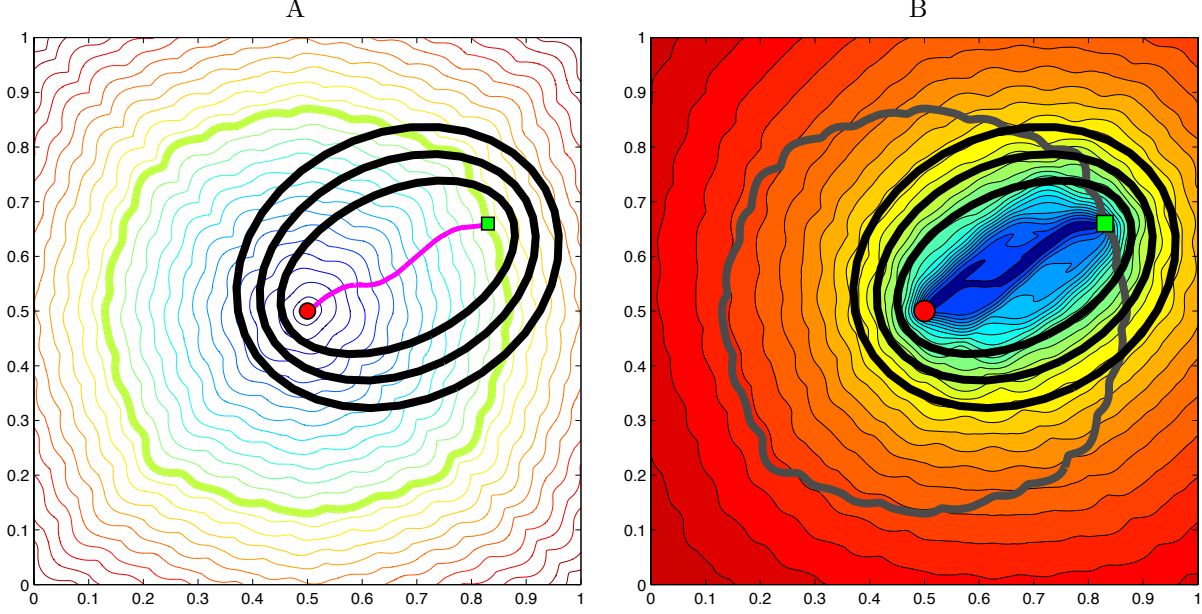


Figure 2: **A.** Level sets of U computed with a highly oscillatory speed $f(x, y) = 2 + 0.5 \sin(20\pi x)(20\pi y)$. The curve ∂L is indicated by a thicker contour line. Three ellipses corresponding to Ψ_1, Ψ_2 , and Ψ_3 are shown in black. **B.** the level sets of $\log_{10}[U(\mathbf{x}) + V(\mathbf{x}) - U(\mathbf{s}) + 0.01]$ for the same problem.

3.2 DYNAMIC DOMAIN RESTRICTION: UNDERESTIMATES AND A^* -TECHNIQUES.

The previous subsection described a priori domain restriction techniques. Here, our goal is to further restrict the computations *dynamically* by using the solution already computed on parts of $\bar{\Omega}$. The actual viscosity solution $u(\mathbf{s})$ depends only on values along a characteristic (i.e., an optimal (\mathbf{s}, \mathbf{t}) trajectory). Ideally, we would like to compute the numerical solution U only for the gridpoints within an immediate neighborhood of that trajectory, potentially yielding a much greater speedup than the techniques described above.

Consider a function $v(\mathbf{x})$ specifying the min-time from \mathbf{s} to \mathbf{x} . (It is easy to see that v is also a viscosity solution of the Eikonal PDE (1.1), but with the different boundary condition $v(\mathbf{s}) = 0$.) We

note that $u(\mathbf{x}) + v(\mathbf{x}) \geq u(\mathbf{s}) = v(\mathbf{t})$ for all $\mathbf{x} \in \Omega$, and this becomes an equality if and only if \mathbf{x} lies on an optimal (\mathbf{s}, \mathbf{t}) trajectory. (See the level sets of $u + v$ in Figure 2B.) Since v is generally unknown, any practical restriction of computational domain will have to rely on an “admissible underestimate heuristic” φ , satisfying $\varphi(\mathbf{x}) \leq v(\mathbf{x})$, $\forall \mathbf{x} \in \bar{\Omega}$. As we will see, tighter underestimates result in more efficient domain restrictions. Here we enumerate several natural heuristic underestimates:

1. *Naïve heuristic* is obtained by assuming the maximum speed of travel along the straight line:

$$\varphi^0(\mathbf{x}) = |\mathbf{s} - \mathbf{x}| / F_2. \quad (3.7)$$

Several papers on SA* versions of FMM [11, 15, 29, 30] have relied on its scaled-down version $\varphi_\lambda^0 = \lambda \varphi^0(\mathbf{x})$ with $\lambda \in [0, 1]$.

2. *Coarse grid heuristic* [16, 18] is based on precomputing V on a coarser $(Rm) \times (Rm)$ grid with $R \in (0, 1)$. If we use V^R to denote the interpolation of that solution on X , the heuristic is then defined as $\varphi_{\lambda,R}^C = \lambda V^R$, where $\lambda \in [0, 1]$ is chosen to ensure that the result is a true underestimate.
3. *Landmarking-based heuristic* [17, 18] is a continuous version of the landmarking technique on graphs [12]. This relies on pre-computing/storing the minimum time from every node to a number of “landmarks”; the triangle inequality is then used to obtain the lower bound $\varphi^L(\mathbf{x}) \leq v(\mathbf{x})$. The high computational cost and memory footprint make this approach useful for repeated queries only. (I.e., only if the optimal trajectory problem has to be solved for many different (\mathbf{s}, \mathbf{t}) pairs.)
4. *Higher-speed heuristic* can be obtained by starting with a special speed-overestimate $f_0(\mathbf{x}) \geq f(\mathbf{x})$, such that the corresponding value function $v_0(\mathbf{x}) \leq v(\mathbf{x})$ is known analytically, and then setting $\varphi(\mathbf{x}) = v_0(\mathbf{x})$. (Note that (3.7) can be also derived this way by taking $f_0(\mathbf{x}) \equiv F_2$.) If the (\mathbf{s}, \mathbf{t}) path-planning has to be performed for *many different* speed functions f , the above approach can be useful even if v_0 has to be approximated numerically. One such example is included in section 4.4.
5. *Scaled “Oracle” heuristic* [18] is defined as $\bar{\varphi}_\lambda(\mathbf{x}) = \lambda v(\mathbf{x})$ with $\lambda \in [0, 1]$. This is clearly not a practical underestimate, but a theoretical device useful in studying the accuracy/efficiency tradeoffs of various domain restriction techniques. Since v is generally unavailable, our benchmarking relies on a numerical approximation; i.e., $\bar{\varphi}_\lambda(\mathbf{x}) = \lambda V(\mathbf{x})$, where V is (pre-)computed on the same grid X .

The first of these (the Naïve heuristic) is a conservative underestimate that is cheaply available for all problems – including the situations with discontinuous speed functions and/or non-convex domains. The other underestimates are more expensive to produce, but usually result in a more significant domain restriction. Thus, their use is particularly justified when the same speed function is used repeatedly to solve numerous (single source / single target) problems.

We emphasize that this paper is in a sense “underestimate-neutral.” A good underestimate is obviously important, but our focus is on how it should be used rather than on how to build it.

Continuous A* Techniques. Both SA* and AA* algorithms on graphs may be easily adapted to the continuous setting using any of the above heuristics. Just like on graphs, SA*-FMM increases the “processing-priority” of nodes with low φ values, whereas our AA*-FMM avoids considering nodes guaranteed not to be a part of any (\mathbf{s}, \mathbf{t}) -optimal trajectory. Each of these methods successfully restricts the computations, but with different trade-offs between the execution time, memory footprint, amount of restriction, and computational error.

3.3 PRIOR WORK ON SA*-FMM.

From the implementation standpoint, SA*-FMM is fairly straightforward. It requires modifying a single line 5 of FMM (see Dijkstra’s algorithm): **ACCEPT** the node with minimal $U + \varphi$. (Since $u + v$ is minimal along the (\mathbf{s}, \mathbf{t}) -optimal trajectory, $U(\mathbf{x}) + \varphi(\mathbf{x})$ is used to indicate how close \mathbf{x} is to that trajectory.) However, the analysis of this method’s output is more subtle.

On graphs, the consistency of the heuristic underestimate (i.e., the condition (2.3)) guarantees that all SA*-accepted nodes receive the same values as would have been produced by Dijkstra's. In contrast, SA*-FMM exhibits a performance tradeoff based on whether φ satisfies a more restrictive and stencil-dependent consistency condition (defined below). If φ is inconsistent, some of the gridpoints may be **ACCEPTED** prematurely, resulting in additional numerical errors. On the other hand, if φ is consistent, the efficiency of the domain restriction is significantly decreased, and this restricted domain does not shrink to zero volume as $h \rightarrow 0$.

The presence of additional errors might seem counterintuitive. After all, if an $(\mathbf{x}_i, \mathbf{t})$ -optimal trajectory passes through some \mathbf{x}_j , then, for φ defined by formula (3.7),

$$u(\mathbf{x}_i) = (\text{Time from } \mathbf{x}_i \text{ to } \mathbf{x}_j) + u(\mathbf{x}_j) \geq \frac{|\mathbf{x}_j - \mathbf{x}_i|}{F_2} + u(\mathbf{x}_j) \geq \varphi(\mathbf{x}_j) - \varphi(\mathbf{x}_i) + u(\mathbf{x}_j),$$

guaranteeing that $u(\mathbf{x}_i) + \varphi(\mathbf{x}_i) \geq u(\mathbf{x}_j) + \varphi(\mathbf{x}_j)$. Turning to numerical solutions, we would hope for the same argument to work for U_i and U_j , and indeed it does if U_i is computed by a one-sided update formula (3.4). But for a first-order upwind discretization in \mathbb{R}^2 , a generic gridpoint \mathbf{x}_i depends on 2 other gridpoints that straddle \mathbf{x}_i 's characteristic. To produce the same numerical values under SA*-FMM and FMM, we would need to know that $U_i + \varphi(\mathbf{x}_i) \geq U_j + \varphi(\mathbf{x}_j)$ whenever \mathbf{x}_i directly depends on \mathbf{x}_j .

Suppose there exists a constant $\lambda > 0$ such that

$$U_i \text{ directly depends on } U_j \implies U_i > U_j + \lambda|\mathbf{x}_i - \mathbf{x}_j|, \quad \forall i, j. \quad (3.8)$$

The proper ordering is then guaranteed provided the underestimate φ satisfies the consistency condition

$$|\varphi(\mathbf{x}_i) - \varphi(\mathbf{x}_j)| \leq \lambda|\mathbf{x}_i - \mathbf{x}_j|, \quad \forall i, j, \quad (3.9)$$

which is easy to ensure by using the underestimate

$$\varphi_\lambda^0(\mathbf{x}) = \lambda\varphi^0(\mathbf{x}).$$

Unfortunately, the condition (3.8) is stencil-dependent and in this section we explore its implications both on grids and triangular meshes; see Figure 3.

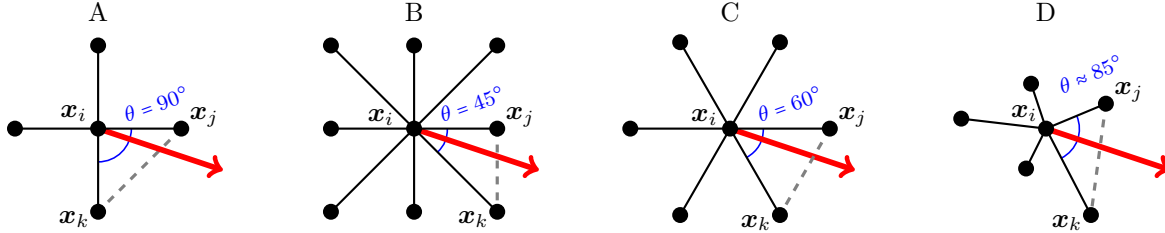


Figure 3: Four computational stencils for a node \mathbf{x}_i : four-point and eight-point stencils on a Cartesian grid (A and B), a six-point stencil on a regular triangular mesh (C), and a five-point stencil on an unstructured triangular mesh (D).

Suppose that \mathbf{x}_i 's characteristic is straddled by \mathbf{x}_j and \mathbf{x}_k , where θ is the angle $\angle \mathbf{x}_j \mathbf{x}_i \mathbf{x}_k$ and γ is the angle between the characteristic and $\mathbf{x}_i \mathbf{x}_j$. Since for the Eikonal equation the characteristics coincide with gradient lines and our numerical approximation is piecewise linear, it is easy to show that¹

$$(U_i - U_j) = \cos \gamma |\mathbf{x}_i - \mathbf{x}_j| / f(\mathbf{x}_i) \geq \cos(\theta) h / f(\mathbf{x}_i).$$

The latter lower bound is actually sharp when the characteristic is parallel to $\mathbf{x}_i \mathbf{x}_k$ and $h = |\mathbf{x}_i - \mathbf{x}_j|$. This means that

¹We note that this observation was previously used in [28] to find the conditions for applicability of Dial-like algorithms.

- $\varphi_0^0 \equiv 0$ is the only consistent underestimate for stencil 3A (i.e., $\lambda = \frac{1}{F_2} \cos \frac{\pi}{2} = 0$). Thus, SA*-FMM will usually result in additional errors.
- for stencil 3B, φ_λ^0 becomes consistent for $\lambda \leq \frac{1}{F_2 \sqrt{2}} = \frac{\cos \frac{\pi}{4}}{F_2}$.
- for a local stencil used on a general triangular mesh (e.g., Figure 3D), if $\bar{\theta} < \frac{\pi}{2}$ is an upper bound on angles θ present in the mesh, then φ_λ^0 becomes consistent for $\lambda \leq \cos(\bar{\theta})/F_2$.

Interestingly, the importance of consistency conditions for SA*-FMM was only recently recognized in [29, 30], while all the prior versions treated this in an ad-hoc fashion. To summarize:

- [2005] Ferguson & Stentz [11] adapt D* algorithms to continuous optimal trajectory problems discretized on stencil 3B. They also introduce an SA*-type technique within D* to further improve the performance. The method relies on φ_λ^0 to ensure the right order of gridpoint processing, but the choice of λ is never explained explicitly.
- [2005, 2006, 2008] Peyré & Cohen [16, 17, 18] adapt SA* for FMM on stencil 3A using underestimates $\varphi_{\lambda,R}^C$ and φ^L . The authors acknowledge that their version of SA*-FMM produces additional errors and experimentally study the dependence of these errors on the tightness of underestimates. However, they do not analyze the behavior of errors under grid refinement.
- [2007] Pêtrès [15] defines an SA*-FMM variant on a stencil 3A with φ_λ^0 . A brief description of a bi-directional version of SA*-FMM is also included. Pêtrès acknowledges that, for large λ , the additional (SA*-induced) errors can be larger than discretization errors, but does not analyze how that ratio changes under grid refinement.
- [2011, 2012] Yershov & LaValle [29, 30] use FMM with acute triangular meshes as in [25] in \mathbb{R}^2 , \mathbb{R}^3 , and on two dimensional manifolds. Their problems of interest use $f \equiv 1$ on a domain with obstacles. The authors use SA*-FMM with φ_λ^0 and prove that $\lambda = \cos(\bar{\theta})$ guarantees absence of additional errors. The authors state that in their experiments SA*-FMM processed only 50% of the gridpoints processed by FMM.

Remark 3.3. Every implementation of SA*-FMM also involves an “efficiency versus memory footprint” tradeoff. Since the binary heap of **CONSIDERED** nodes is sorted based on $U + \varphi$, every heap-maintenance operation relies on availability of $\varphi(\mathbf{x})$ for many nodes on the heap. This happens whenever a **FAR** node becomes **CONSIDERED**, or a **CONSIDERED** node receives a smaller value or becomes **ACCEPTED**. If φ is re-computed each time it is needed (e.g., by (3.7)), this introduces a noticeable overhead to each heap operation. An alternative (to cache φ the first time it is computed for each **CONSIDERED** node) is certainly more efficient, but significantly increases the memory footprint, particularly on larger grids and in higher-dimensional problems. In Section 4 we include the performance data for both of these approaches.

3.4 ACCURACY OR EFFICIENCY?

The errors introduced by any A*-type restriction techniques are not very surprising once we recall that the numerical viscosity of the discretization results in a large domain of computational dependency for $U(\mathbf{s})$. To formalize this argument, we will consider a *dependency digraph* G built on the nodes of X . For \mathbf{x}_i and $\mathbf{x}_j \in N_i$, G includes an arc $(\mathbf{x}_i, \mathbf{x}_j)$ if \mathbf{x}_i *directly depends* on \mathbf{x}_j ; i.e., if U_j is needed to compute U_i . We will say that \mathbf{x}_i *depends* on \mathbf{x}_j if there exists a path in G from \mathbf{x}_i to \mathbf{x}_j . Due to the monotone causality of the upwinding discretization, this dependence implies $U_i > U_j$; thus, G is acyclic and every path on it leads to \mathbf{t} . We will also use $G(\mathbf{s})$ to denote the subset of G reachable from \mathbf{s} .

Consider *any* domain restriction technique that results in accepting only nodes from some $\hat{X} \subset X$ and produces some numerical approximation of the value function $U^*(\mathbf{x})$ for each $\mathbf{x} \in \hat{X}$. If $G(\mathbf{s}) \not\subset \hat{X}$, we cannot expect $U^*(\mathbf{s})$ to be the same as $U(\mathbf{s})$ produced by FMM on the full X . In other words, if we insist on avoiding *any* additional (restriction-induced) errors, this typically results in severe

constraints on the efficiency of the domain restriction.

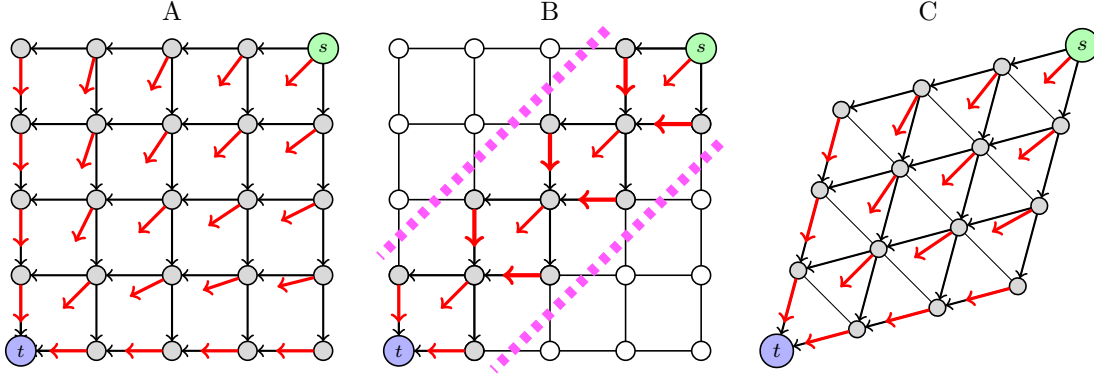


Figure 4: Domain restriction for the constant speed example. The full dependency graph is shown for a 4-point stencil on a cartesian grid (A) and for a 6-point stencil on a triangular mesh (C). Thin black arrows show the arcs of $G(s)$. The shorter arrows show the characteristic direction for each node. Subfigure (B) shows a domain-restricted computation. The nodes inside of the dashed lines represent the nodes that pass the A^* condition (3.10). The thicker characteristic arrows highlight the “optimal” directions that have changed due to this domain restriction.

To illustrate this point, we consider a very simple problem with t and s in opposite corners of $\bar{\Omega}$; see Figure 4A. With $f \equiv 1$ the optimal trajectory from every starting position x is just a straight line to t . But it is easy to see that $G(s)$ includes all nodes in X ; thus, **any** restriction will result in $U^*(s) > U(s)$. We emphasize that this phenomenon has nothing to do with the non-existence of consistent φ for the 4-point stencil discretization on a cartesian grid. Figure 4C shows an equivalent example on a regular triangular mesh. As explained in [29, 30], taking $\lambda = 1/2$ will ensure that φ_λ^0 is consistent for this problem and stencil. As a result, SA^* -FMM will produce $U^*(s) = U(s)$, but at the cost of accepting exactly the same set of nodes² as FMM (i.e., $\hat{X} = X$).

For these reasons, we believe that asking for $U^*(s) = U(s)$ on every fixed grid is unrealistic and makes the domain restriction much less efficient. A more attractive strategy is to ensure that $|U^*(s) - U(s)|$ is small relative to discretization errors and $U^*(s) \rightarrow u(s)$ as $h \rightarrow 0$. This can be ensured provided \hat{X} covers a neighborhood of the (s, t) -optimal trajectory **and** $U^* = \hat{U}$, the solution that FMM would have produced on \hat{X} . This is precisely what AA^* -FMM does when used with an inconsistent φ ; on the other hand, the different order of acceptance under SA^* -FMM typically results in $U^* \neq \hat{U}$ and a lack of convergence (or a very slow convergence – see Section 4.1) under grid refinement.

3.5 THE NEW METHOD: AA^* -FMM

The AA^* technique is also quite easy to use in the continuous setting as a modification of the standard FMM. Our current implementation is based on the upwind discretization (3.2) on a standard 4-point stencil, but the required FMM-changes would be the same for any other monotone-causal stencil (either on a grid or on a simplicial mesh). Similarly to a version of AA^* for graphs:

- we rely on an overestimate Ψ of the time along the (s, t) -optimal trajectory (see section 3.1);
- the **CONSIDERED** nodes are still sorted by U -values, and thus the underestimate φ does not have to satisfy any consistency conditions;
- we only mark a node x as **CONSIDERED** if it satisfies the “ A^* condition”

$$U(x) + \varphi(x) \leq \Psi. \quad (3.10)$$

² The computational savings of 50% were reported in [29, 30] for $f \equiv 1$ on the domain with obstacles. Based on the above discussion, such savings are in fact highly dependent on the size of $G(s)$ relative to the total number of meshpoints. This percentage is, in turn, defined by the type of the mesh and the positions of s and t relative to the obstacles.

This simple criterion allows for AA* to be adapted to *both* label-setting and label-correcting methods. If φ satisfies the consistency condition (3.9), the values produced by AA*-FMM are also the same as those resulting from FMM, but on a smaller (**ACCEPTED**) subset of the grid \hat{X} . However, AA*-FMM can be also used even if φ does not satisfy (3.9), which results in additional errors but does not prevent the convergence to viscosity solution of the PDE under grid refinement.

To illustrate the efficiency of the AA*-type domain restrictions, we consider the boundaries of 3 sets:

$$\begin{aligned} C_1 &= \{\mathbf{x} \mid |\mathbf{x} - \mathbf{s}| + |\mathbf{x} - \mathbf{t}| \leq F_2 \Psi\}, \\ C_2 &= \{\mathbf{x} \mid u(\mathbf{x}) + \varphi(\mathbf{x}) \leq \Psi\} \subseteq C_1, \\ C_3 &= \{\mathbf{x} \mid u(\mathbf{x}) + v(\mathbf{x}) \leq \Psi\} \subseteq C_2. \end{aligned} \quad (3.11)$$

All three are shown in Figure 5 for the example introduced in section 3.1. Both u and v are numerically approximated by FMM on the entire domain Ω . The boundaries ∂C_i are shown by bold lines for Ψ_1, Ψ_2 , and Ψ_3 . The set C_1 corresponds to the ellipse defined for each specific Ψ . The set $C_2 \cap L$ is roughly the set accepted by AA*-FMM with the specified Ψ **and** the underestimate φ^0 . The set $C_3 \cap L$ is the minimum part of the domain that AA*-FMM would have to accept with that Ψ even if we were to use the perfect $\varphi = \bar{\varphi}_1$. If $\Psi = \Psi_3$, then C_3 collapses to the optimal trajectory.

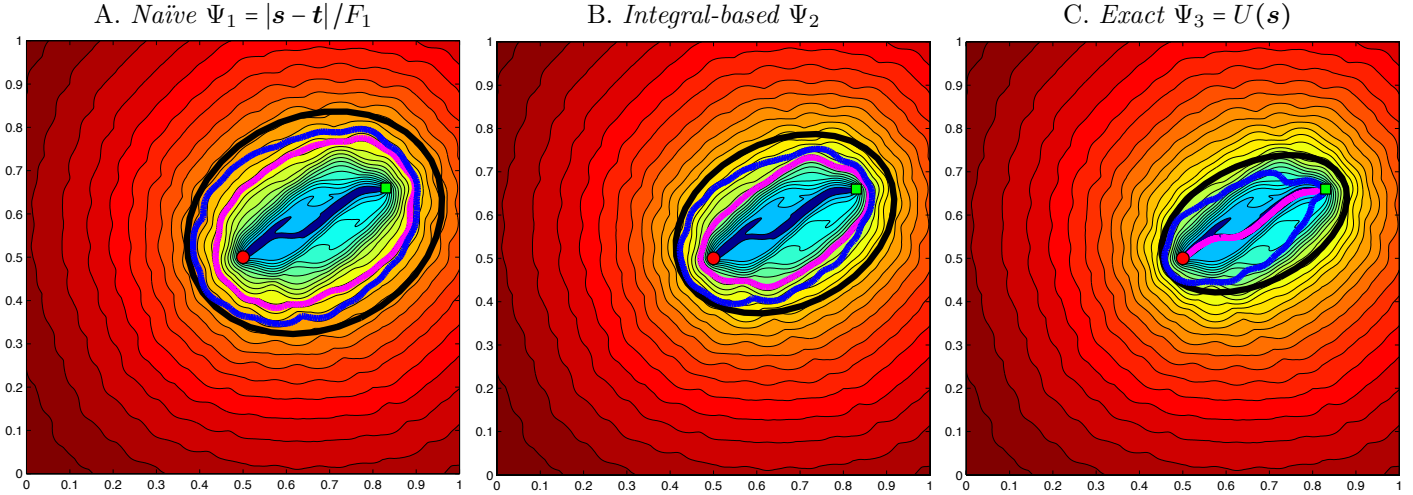


Figure 5: Level sets for $u+v$ computed by FMM on a 401^2 grid. In each subfigure, the bold lines show the boundaries of C_1 , C_2 , and C_3 (from out-to-in) for the specified Ψ .

This Figure also clearly demonstrates the importance of an accurate Ψ for the efficiency of the domain restriction in AA*-FMM. If the initial Ψ is not particularly tight, the performance can be further improved by decreasing Ψ *dynamically* in a Branch&Bound fashion. This approach relies on availability of a “*heuristic overestimate*” $\psi(\mathbf{x}) \geq v(\mathbf{x})$, $\forall \mathbf{x} \in \bar{\Omega}$. For a convex domain, our implementation uses an obvious (and cheaply computed) overestimate

$$\psi(\mathbf{x}) = |\mathbf{x} - \mathbf{s}| / F_1, \quad (3.12)$$

which is consistent with our definition of Ψ_1 in §3.1. Each time a gridpoint \mathbf{x} is **ACCEPTED**, this AA*BB-FMM algorithm attempts to decrease Ψ as follows:

$$\Psi \leftarrow \min \{\Psi, U(\mathbf{x}) + \psi(\mathbf{x})\}. \quad (3.13)$$

A better ψ can be obtained by numerically integrating the slowness along any feasible (\mathbf{s}, \mathbf{t}) trajectory, or even using PMP-based techniques. Performing such computations for every **ACCEPTED** gridpoint would be clearly prohibitive, but using it every so often (in addition to the systematic use of formula (3.12)) could be a useful technique to investigate in the future.

Remark 3.4. On graphs, using the exact “underestimate” $\varphi = V$ simply resulted in accepting only those nodes that lie on the optimal path. In the continuous case, the optimal trajectory does not pass through every node it directly depends on. Even for a node \mathbf{x} immediately next to the optimal (\mathbf{s}, \mathbf{t}) trajectory, if the underestimate φ is very accurate, this may cause the A* condition (3.10) to fail (resulting in \mathbf{x} never becoming **CONSIDERED**). This situation rarely arises in practice – e.g., with $\varphi = \varphi^0$ this can happen only if Ψ is exact and the speed $f(\mathbf{x}) = F_2$ on some neighborhood of \mathbf{s} . We have used two different approaches to address this issue:

- Introduce a numerical tolerance factor; i.e., use $(1 + \epsilon_{tol} h^\mu) \Psi$ instead of Ψ . Our analysis of restriction-caused errors in Section 6 applies as long as $\epsilon_{tol} > 0$ and $\mu \in [0, 1/2)$. All the numerical tests in Section 4 rely on this approach and confirm the convergence even with $\mu = 1/2$.
- Alternatively, if \mathbf{s} has not been accepted by the end of AA*-FMM, one can simply take $U(\mathbf{s}) = \Psi$. Since Ψ was obtained as a cost of some known (\mathbf{s}, \mathbf{t}) trajectory, that trajectory is then declared optimal (at least for the current grid resolution).

4 NUMERICAL RESULTS

All algorithms were implemented in C++ and compiled with **g++** version 4.2 on a Macbook Pro (4 GB RAM and an Intel Core i7 processor – four 2 GHz cores). To make the benchmarking results as compiler/platform-independent as possible, we have turned off all compiler optimizations (option -O0). For all of the 2D and 3D examples, $\bar{\Omega} = [0, 1]^n$ is discretized by a uniform cartesian grid with m^n gridpoints. To test the numerical approximation errors in distance computations (Section 4.1), we have used an analytical solution $u(\mathbf{x}) = |\mathbf{x} - \mathbf{t}|$. In all other cases, the ‘ground truth’ u was computed numerically by FMM on the full domain using the ‘highly’ refined grid:

Dimension	Ground truth	Resolutions considered
$n = 2$	$m = 6401$	$m = 101, 201, 401, 801, 1601$ and 3201
$n = 3$	$m = 401$	$m = 26, 51, 101, 201$ (and 401 when $f \equiv 1$)

Accuracy metrics. Since we are interested in single-source / single-target problems, all accuracy metrics are based on comparing various numerical approximations and the true solution at a single point \mathbf{s} . As before, we use U to denote the solution produced by FMM on the entire X while U^* denotes the solutions produced by the respective A*-modifications of FMM. We base our comparison on the following “relative errors” for each example:

$$\begin{aligned}
\mathcal{E}^d &= \text{relative discretization error (DE) at } \mathbf{s} \text{ using FMM} &= |U(\mathbf{s}) - u(\mathbf{s})| / u(\mathbf{s}) \\
\mathcal{E}^* &= \text{relative error at } \mathbf{s} \text{ when using A}^* &= |U^*(\mathbf{s}) - u(\mathbf{s})| / u(\mathbf{s}) \\
\mathcal{E}_N^* &= \text{relative error at } \mathbf{s} \text{ explicitly due to A}^* &= [U^*(\mathbf{s}) - U(\mathbf{s})] / U(\mathbf{s}) \geq 0.
\end{aligned}$$

Since the upwind discretization is convergent, $U \rightarrow u$ and thus $\mathcal{E}^d \rightarrow 0$ as $h \rightarrow 0$. Correspondingly, a successful domain restriction should have $\mathcal{E}^* \rightarrow 0$ (and thus $\mathcal{E}_N^* \rightarrow 0$) as $h \rightarrow 0$.

To measure the efficiency of the domain restriction, we also define

$$\mathcal{P} = \text{fraction of domain computed} = (\# \text{ of gridpoints } \text{ACCEPTED} \text{ or } \text{CONSIDERED}) / m^n.$$

Underestimate functions. In all examples except for section 4.4, we rely on naïve and scaled-oracle heuristics (i.e., φ_λ^0 and $\bar{\varphi}_\lambda$). We consider this sufficient since the accuracy of the AA* approach is really underestimate-neutral (though the efficiency is clearly dependent on both φ and Ψ). We expect that the results based on any other heuristics (including those in [16, 17, 18]) will be qualitatively similar.

4.1 CONSTANT SPEED $f \equiv 1$ IN 2D AND 3D

In the constant speed case, all characteristics are straight lines and the naïve heuristic coincides with the actual time-to-go (i.e., $\varphi^0 = v$). In this subsection we use the underestimate $\varphi = \varphi_\lambda^0$

and place \mathbf{s} and \mathbf{t} at opposite corners of $\bar{\Omega}$. Our goal is to test the effect of $\lambda \in [0, 1]$ on the accuracy and efficiency for different grid resolutions $h = 1/(m - 1)$. In testing AA*-FMM, we use $\Psi = (1 + \epsilon_{tol} h^\mu)|\mathbf{s} - \mathbf{t}|$, where $\mu = 1/2$ with $\epsilon_{tol} = 1/4$ in 2D and $\epsilon_{tol} = 1/3$ in 3D. This ensures that AA*-FMM does not terminate before \mathbf{s} is **ACCEPTED** and also results in the set $C_3 = C_2$ shrinking to a straight line as $h \rightarrow 0$.

Figure 6 shows the level sets of U^* computed by SA*-FMM and AA*-FMM on a 2D grid with $m = 351$ and $\lambda \in \{0.25, 0.5, 0.75, 1\}$. The non-smoothness of the level-sets produced by SA*-FMM is due to the additional errors introduced by that method. For $\lambda = 1$ these errors also result in a larger \mathcal{P} – despite the fact that our AA*-FMM has a built-in “restriction slackness” (since $\Psi > u(\mathbf{s})$). Figure 7 shows $\log_{10}(\mathcal{E}_N^*)$ as m and λ vary. For $\lambda \geq 0.55$, the errors produced by SA*-FMM are not only relatively large, but also do not decrease much under grid refinement. In contrast, the errors in AA*-FMM decrease quite rapidly even though the set C_3 is also shrinking as $h \rightarrow 0$; see also the convergence analysis in Section 6.

Since in this example $G(\mathbf{s}) = X$, additional errors should result from *any* domain restriction. However, the finite-precision of the floating point arithmetic results in “zero domain restriction errors” (white spaces in Figure 7) for AA*-FMM even for many test runs where $G(\mathbf{s})$ is partly truncated. E.g., see the case $(\lambda = 0.75, m = 351)$ in Figures 6 and 7.

Figures 8 and 9 show the full accuracy/efficiency data holding $\lambda = 1$ and varying m .

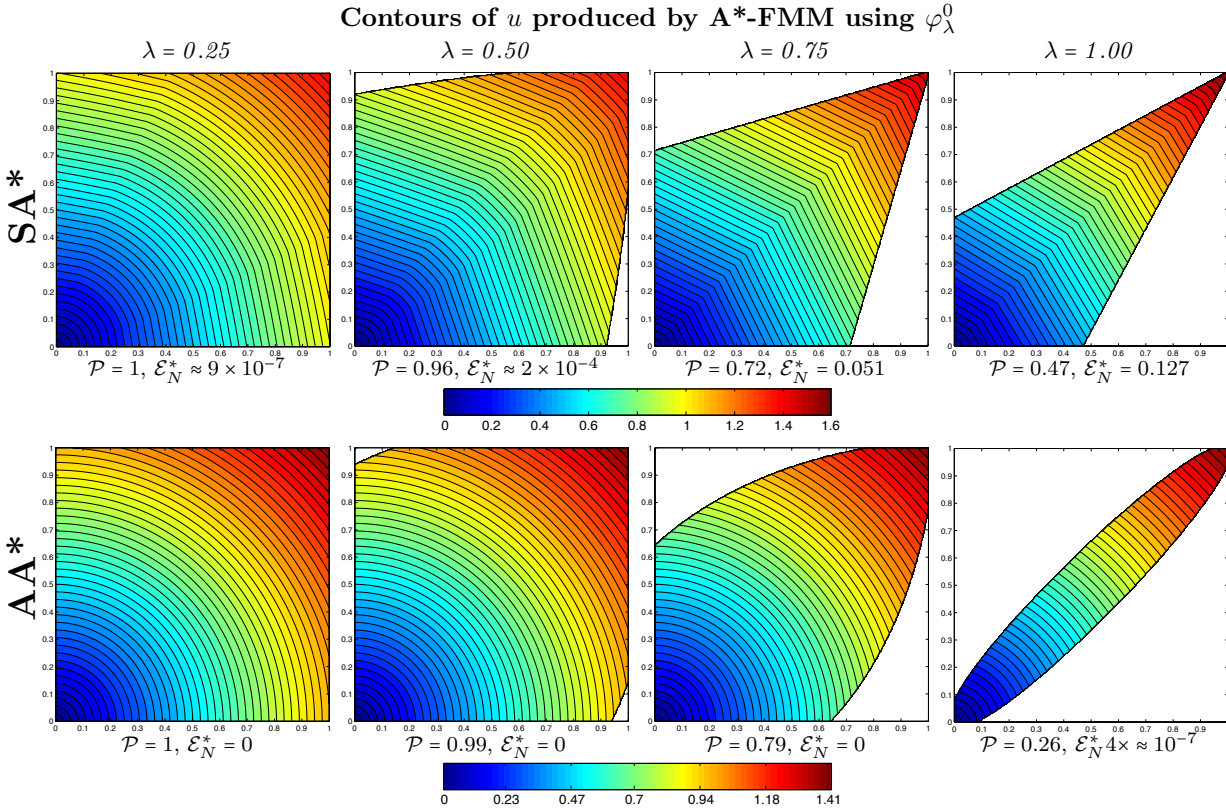


Figure 6: The top row was produced with SA*-FMM and the error can be seen in two ways: (1) the deformation of the level sets and (2) the value at the source is ≈ 1.61 . The bottom row shows the results of AA*-FMM. We hold $m = 351$ while λ values increase from left to right.

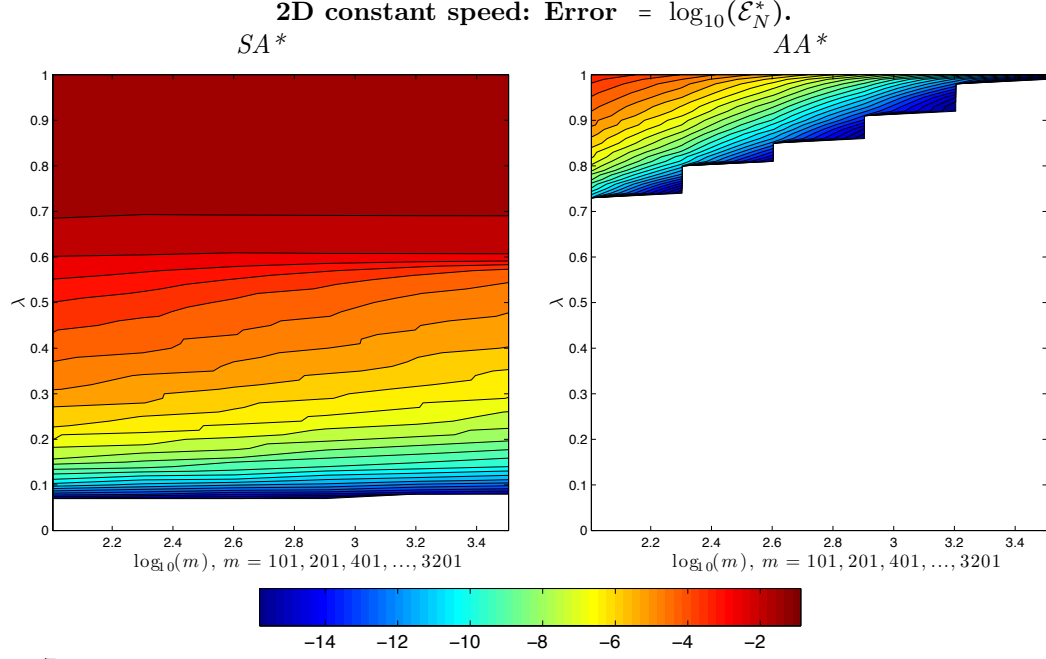


Figure 7: SA^* versus AA^* comparison based on \mathcal{E}_N^* errors. The horizontal axis shows the grid resolution m , and the vertical axis corresponds to the heuristic strength λ . White corresponds to errors smaller than the machine ε .

2D constant speed: Statistics.

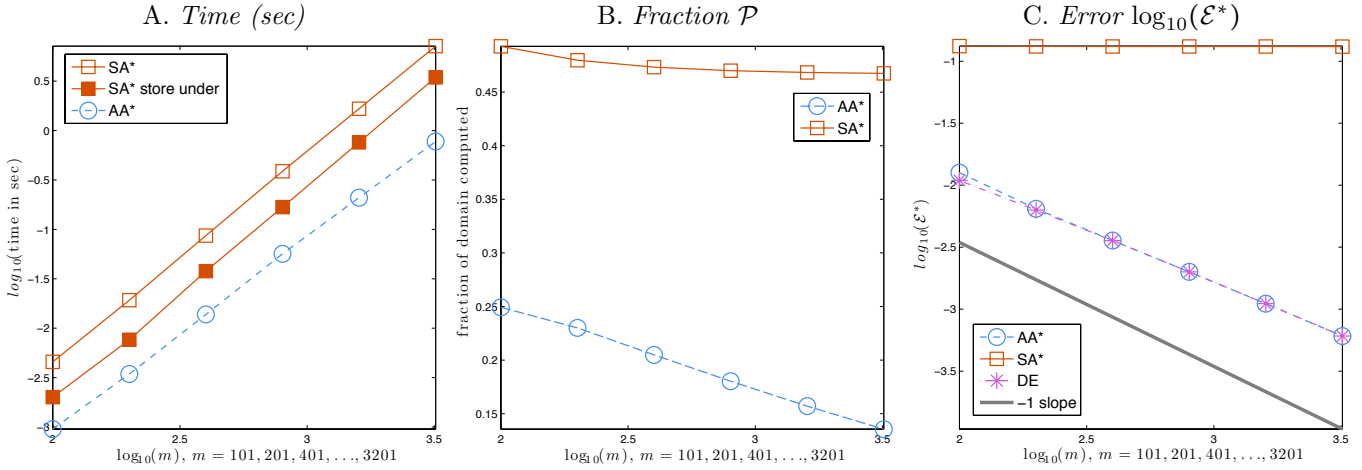


Figure 8: The CPU-time, the fraction \mathcal{P} of the domain computed, and the error \mathcal{E}^* for both SA^* and AA^* using a constant speed function in 2D. The solid square markers in the time plot indicate the time for a version of SA^* -FMM that stores each $\varphi(\mathbf{x})$ after it is first computed. The underestimate function used is φ^0 and The benchmarking is performed for $\lambda = 1$ (i.e., corresponding to the very top slice in Figure 7).

3D constant speed: Statistics.

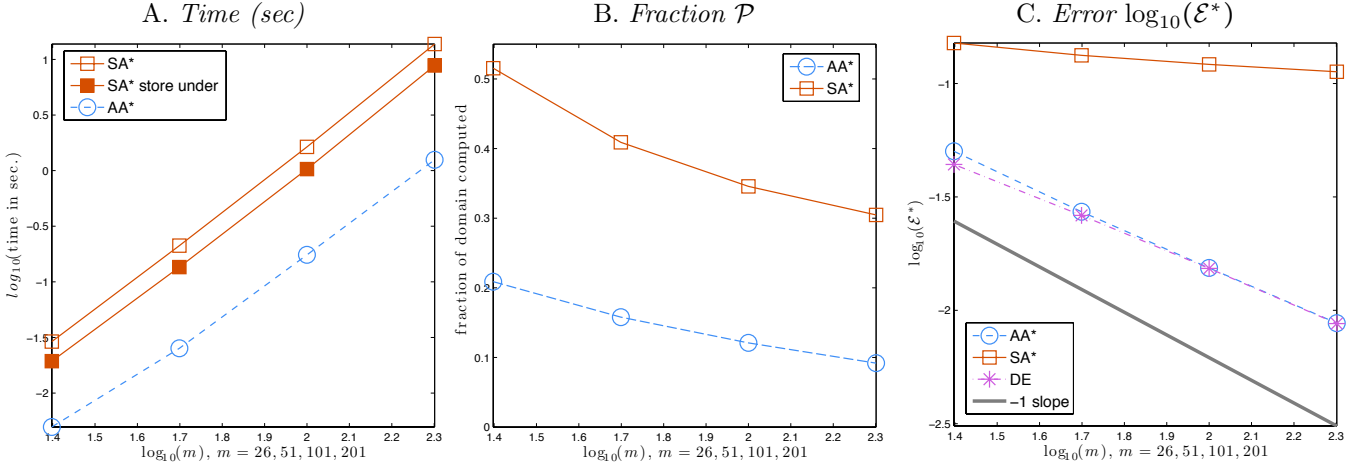


Figure 9: The same data as in Figure 8, but for 3D computations.

4.2 OSCILLATORY SPEED FUNCTION IN 2D AND 3D

For the next 2D example, we set $\mathbf{s} = (0.95, 0.7)$ and $\mathbf{t} = (0.5, 0.5)$ and consider a highly oscillatory speed

$$f(x, y) = 1 + 0.5 \sin(20\pi x) \sin(20\pi y), \quad (4.1)$$

resulting in frequent directional changes along most optimal paths. We start by focusing on a scaled oracle heuristic $\varphi = \bar{\varphi}_\lambda$ with AA*-FMM also relying on $\Psi = (1 + \epsilon_{tol} h^\mu) v(\mathbf{s})$. Figure 10 shows the level sets of numerical solutions obtained with $m = 401$. We note that the SA*-errors result in a significant distortion of the optimal trajectory (see the switch between $\lambda = 0.3$ and $\lambda = 0.7$).

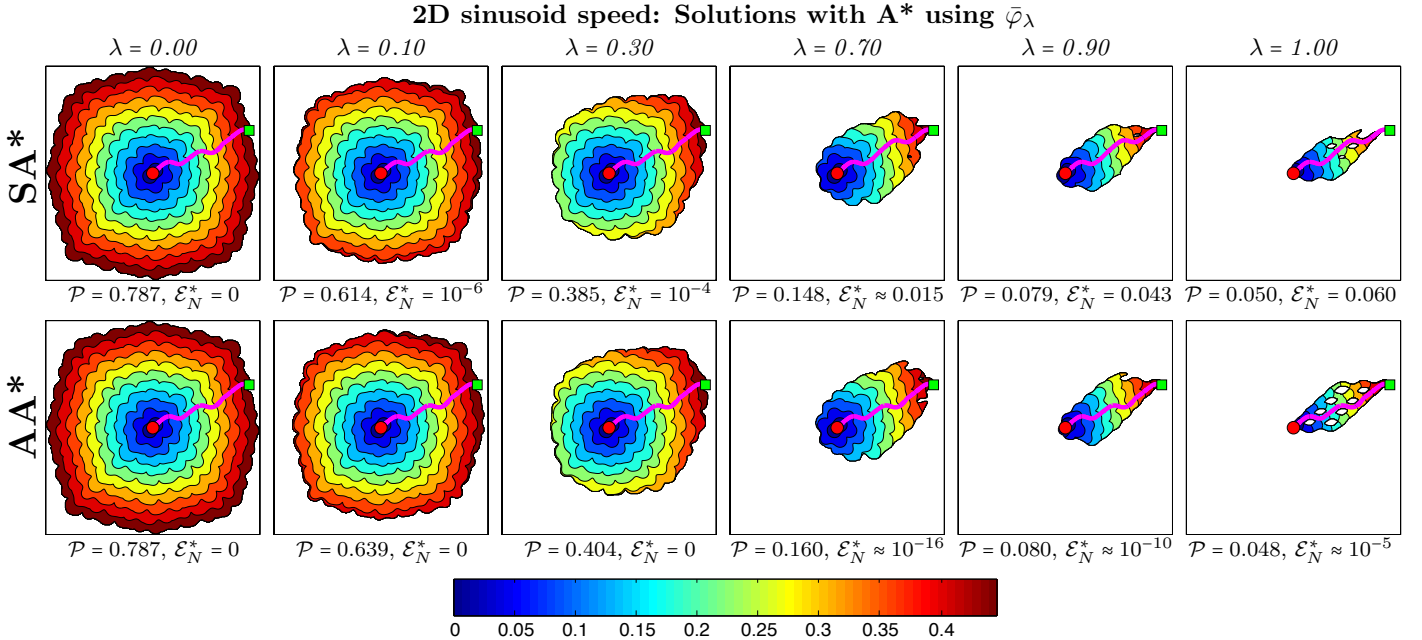


Figure 10: Numerical results of FMM combined with SA* and AA*, showing the fraction of domain computed \mathcal{P} and the relative error \mathcal{E}_N^* . Note the change in the ‘optimal’ trajectory for SA* between $\lambda = 0.3$ and $\lambda = 0.7$. The solutions were produced using $m = 401$.

Figure 11 compares the accuracy of these techniques for different (m, λ) pairs. Qualitatively the picture is largely the same as in Figure 7, but with two non-trivial differences. First, the ‘white

block' in the lower-left corner of the SA* plot indicates the lack of additional errors with $m = 101$ and $\lambda \leq 0.15$. Based on our computational experiments, this is an *extremely* rare situation – the only example we could find, where the entire $G(\mathbf{s})$ is processed by SA*-FMM in the correct order despite the fact that the heuristic φ is inconsistent. Second, we observe that the AA*-FMM-generated errors are not always monotone decreasing in m . E.g., the errors are present for $(m = 401, \lambda = 0.75)$, but not for $(m = 201, \lambda = 0.75)$, where the entire $G(\mathbf{s})$ is accepted.

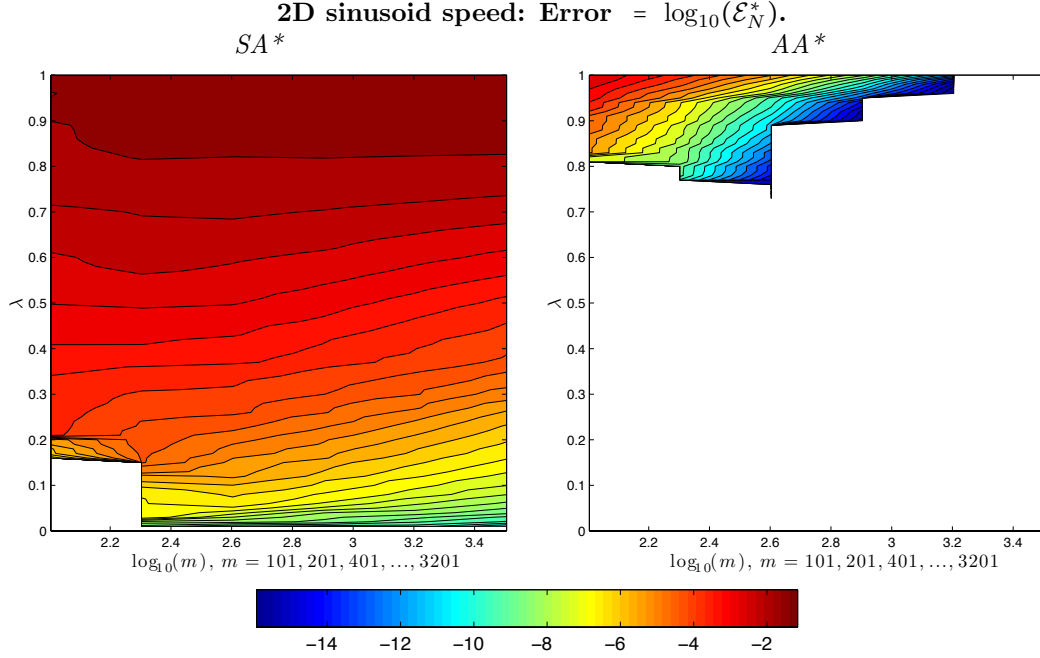


Figure 11: This plot shows the same results as Figure 7 except with the sinusoid speed (4.1).

Since the oracle heuristic is generally unavailable, we now consider the accuracy/efficiency tradeoffs using the naïve heuristic $\varphi = \varphi^0$ and a realistically obtainable (but conservative) overestimate $\Psi = \Psi_2$. Figure 12 shows that AA*-FMM yields comparable efficiency (despite accepting a larger part of the domain) while also ensuring $U^*(\mathbf{s}) = U(\mathbf{s})$ since the entire $G(\mathbf{s})$ is accepted.

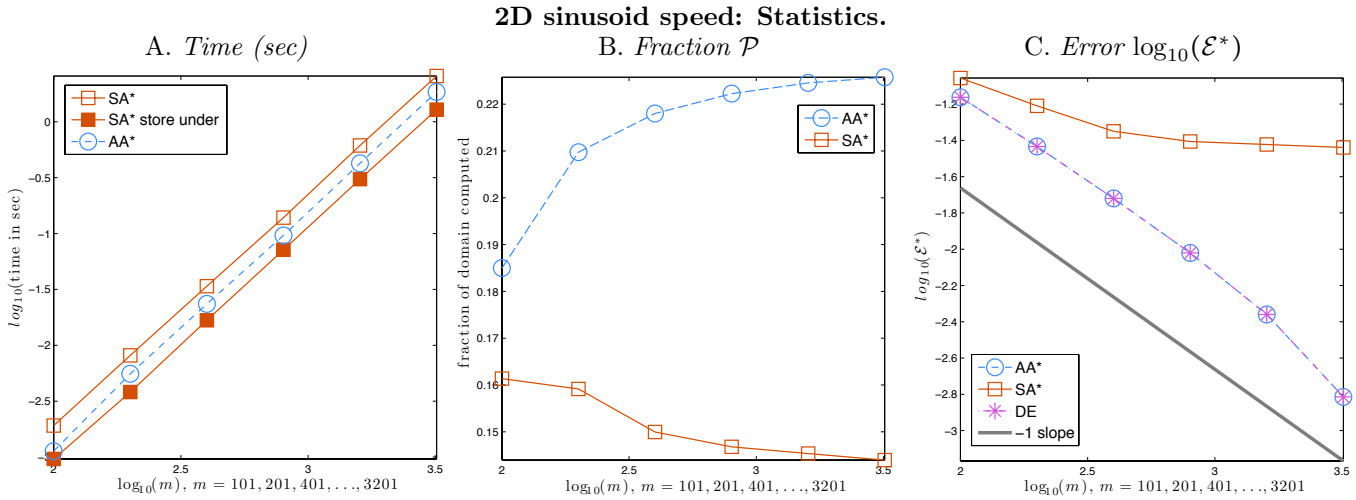


Figure 12: These results show the time (in seconds), fraction domain calculated, and the error \mathcal{E}^* for both SA* and AA* using a highly oscillatory sinusoid function in 2D. The naïve heuristic was used, and for AA* $\Psi = \Psi_2$.

We also consider similar oscillatory examples in 3D with

$$f(x, y, z) = 1 + A \sin(10\pi x) \sin(10\pi y) \sin(10\pi z), \quad (4.2)$$

for two amplitudes $A = 0.1$ and $A = 0.35$. The source/target locations are $\mathbf{s} = (0.72, 0.6, 0.8)$ and $\mathbf{t} = (0.32, 0.4, 0.36)$. Figure 13 shows the accuracy/efficiency data based on realistic $\varphi = \varphi^0$ and $\Psi = \Psi_2$. The errors due to AA* are negligible compared to discretization errors, while the errors due to SA* are again quite noticeable and decrease much slower as $h \rightarrow 0$.

3D sinusoid speed: Statistics.

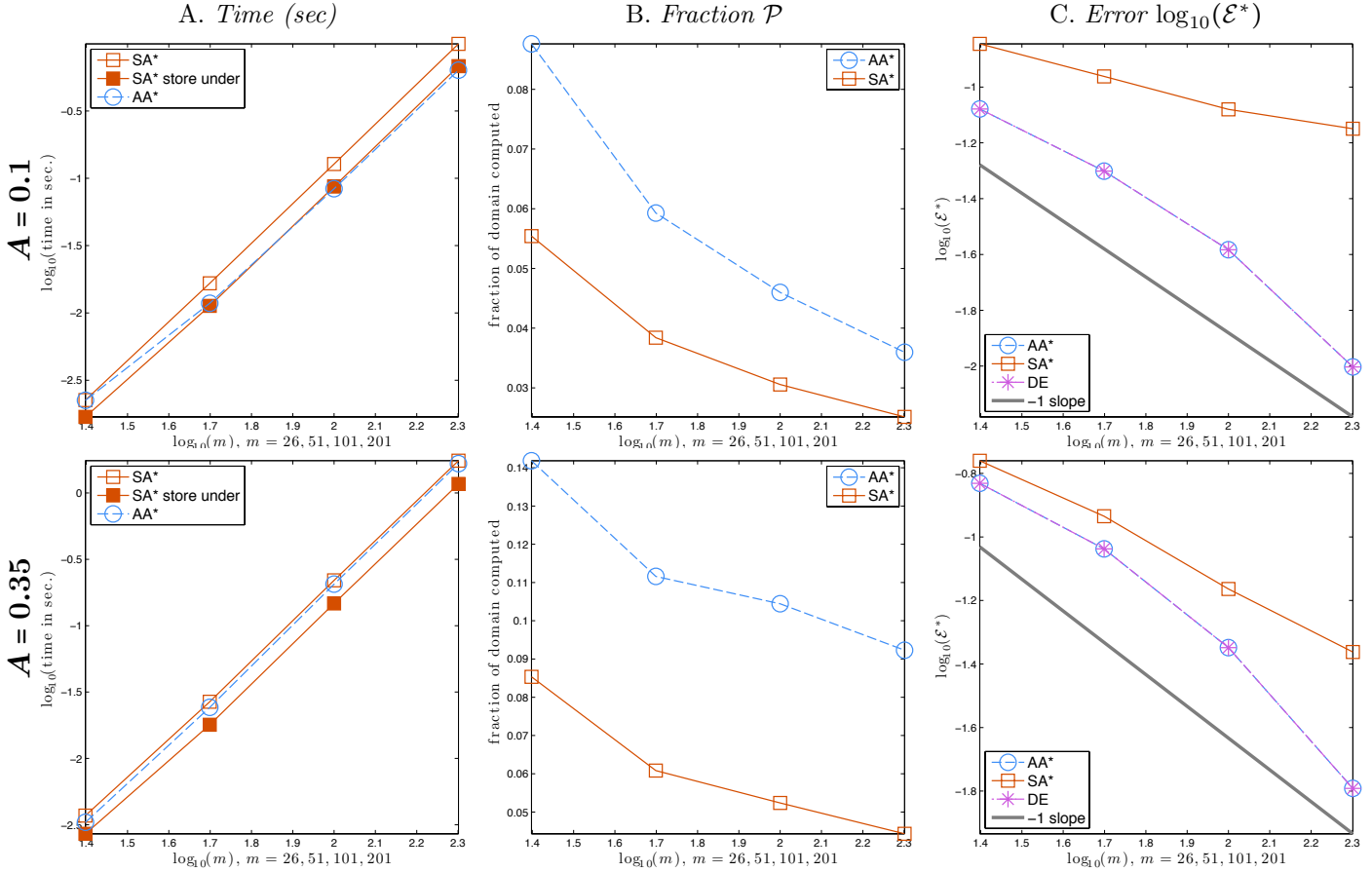


Figure 13: These results again show the average time (in seconds) of 10 trial runs, fraction domain calculated, and the error \mathcal{E}^* for both SA* and AA* using (4.2). The naïve heuristic was used, and for AA* $\Psi = \Psi_2$. The top row corresponds to $A = 0.1$ and the bottom row shows the results when $A = 0.35$. When $A = 0.1$ the result might seem counterintuitive: AA* takes less CPU time even though it processes more of the domain. Careful profiling shows that for SA* the three-neighbor update fails more frequently and causes the algorithm to perform more two-sided updates. This makes an average node update in SA* more computationally expensive; hence the slower time.

4.3 SATELLITE IMAGE

The following path-planning example is borrowed from [16, 17, 18]. The grayscale intensities of a satellite photograph (Figure 14A) are imported into the range $[0, 755]$ using MATLAB's `imread()` routine. For a given gridpoint \mathbf{x} assume that it falls into a pixel with grayscale value $i(\mathbf{x}) \in [0, 755]$. This then defines the speed $f: \bar{\Omega} \rightarrow [0.001, 1.001]$ via rescaling:

$$f(\mathbf{x}) = 0.001 + i(\mathbf{x})/755$$

This is the same intensity/speed mapping used in [18], but our experimental setup is slightly different:

- Unlike Peyré et al., we omit the pre-smoothing of the original 744×744 image and simply downsample it to 350×350 .
- Peyré et al. use $\varphi = \varphi_{\lambda,R}^C$; they fix $\lambda = \frac{1}{2}$ and vary R . Instead, we first use $\varphi = \varphi^0$ (Figure 14B) and then switch to $\varphi = \bar{\varphi}_\lambda$ (Figure 15). Unlike with $\varphi_{\lambda,R}^C$, the use of $\bar{\varphi}_\lambda$ directly illustrates the performance of A* as the quality of φ improves.
- We use slightly different source and target locations ($\mathbf{s} = (337h, 161h)$ and $\mathbf{t} = (16h, 188h)$; see Figure 14C). Our \mathbf{s} falls on the opposite side of a shockline compared to \mathbf{s} used in [18].

Figure 14B shows the level sets of the solution on the full domain, with ∂L and ∂C_2 (using φ^0 and Ψ_3) shown in bold. (The set **ACCEPTED** by SA* is approximately the same as AA*.)

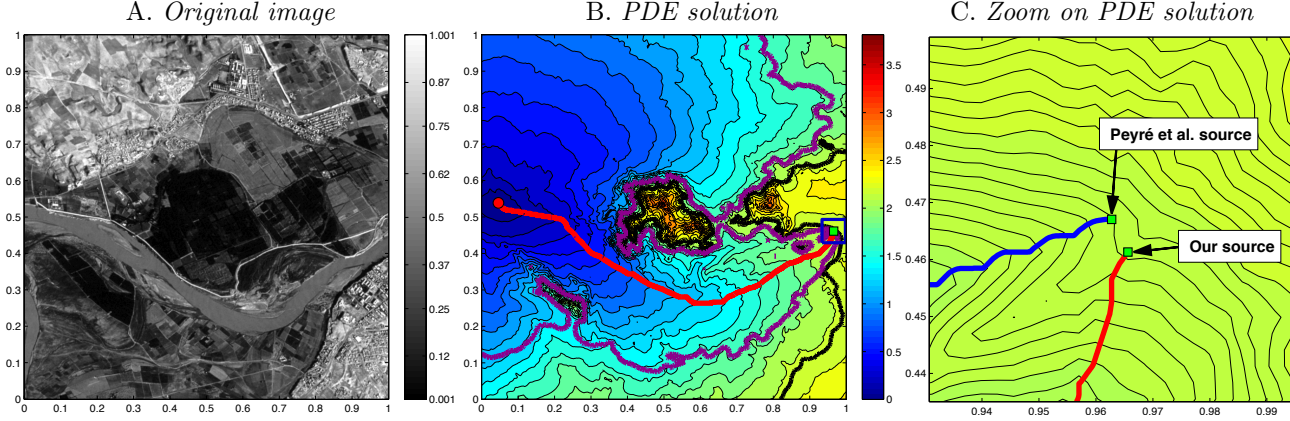


Figure 14: A. The original satellite image mapped to a speed $f \in [0.001, 1.001]$. B. The solution to the PDE on a 350×350 grid with ∂L and ∂C_2 (using φ^0 and Ψ_3) drawn in bold. C. The upper marker is approximately the same \mathbf{s} as in [18]; the lower marker is the same \mathbf{s} used in B and Figure 15A.

This example illustrates the use of A*-techniques with a discontinuous speed function. The rather limited computational savings in 14B are clearly caused by the use of an “overly optimistic” φ^0 . However, the lack of accuracy of this naive underestimate is not caused by any discontinuities in f – instead it is simply a result of a large F_2/F_1 , with f values much closer to F_1 on most of the domain.

We now switch to “oracle tests” with $\varphi = \bar{\varphi}_\lambda$ and AA*-FMM relying on $\Psi = (1 + \sqrt{h}/8)V(\mathbf{t})$. Using this heuristic, the domain restriction becomes much more effective for both SA* and AA*. But since \mathbf{s} is close to a shockline, additional errors due to SA*-FMM are sufficiently large to change the optimal trajectory in several ways (see Figure 15A). In contrast, the errors from AA*-FMM are much smaller and the optimal trajectory remains the same for all λ .

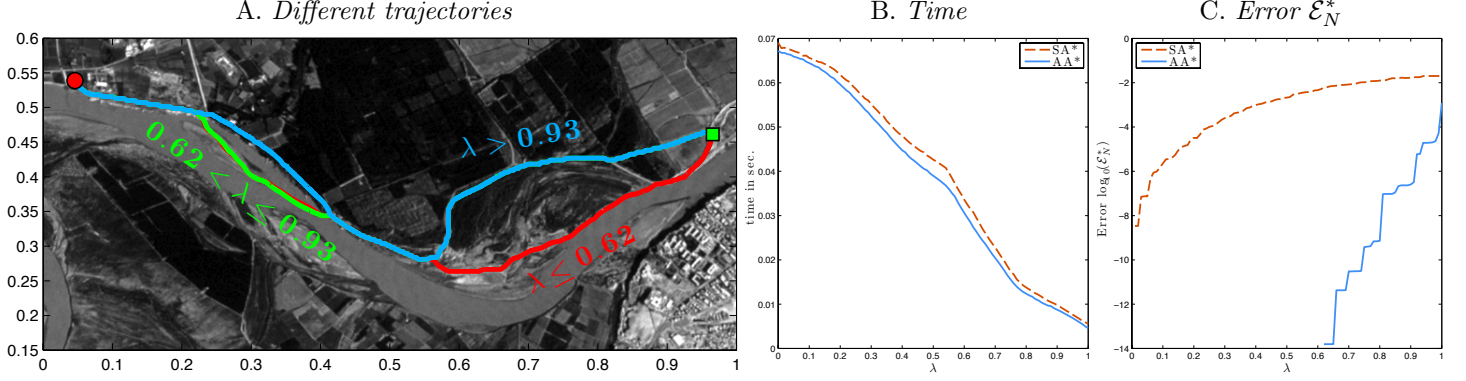


Figure 15: A. The λ -dependent “optimal” trajectories recovered by SA*-FMM (red, green, and blue curves). AA*-FMM always recovers the truly optimal (red) trajectory. When the trajectories overlap, the red red curve lies under the green, and the green curve lies under the blue. B&C. The time (in seconds) and \mathcal{E}_N^* produced by SA* and AA* as λ changes in $[0, 1]$.

4.4 REPLANNING IN A DYNAMIC ENVIRONMENT

Our final example illustrates several important points:

1. The use of special/custom underestimate φ based on a related control problem.
2. The optimal trajectory from a related control problem is valuable as an initial guess for the Pontryagin Maximum Principle (PMP).
3. The PMP-computed trajectory is not necessarily globally optimal, but can be used to produce an accurate Ψ .

Here we will use a slightly more general setup where the task is to minimize the total *cost* (instead of considering only the *time*) to reach t . Given a running cost function $K : \Omega \rightarrow (0, +\infty)$ integrated along the trajectory and a speed f_0 , the value function u now satisfies a different Eikonal PDE given by

$$|\nabla u(\mathbf{x})| f_0(\mathbf{x}) = K(\mathbf{x}). \quad (4.3)$$

Our specific problem is to find the “safest” trajectory in an adversarial environment, with K higher on the parts of the domain more closely monitored by the adversary.

If we assume no prior information on enemy locations and monitoring patterns, it is natural to select $K \equiv 1$, which implies that the quickest trajectory is in fact the safest. Consider the domain $\bar{\Omega} = [-0.05, 0.85] \times [0, 0.9]$ with the speed and running-cost defined by

$$f_0(x, y) = 1 + 0.99 \sin(4\pi x) \sin(4\pi y) \quad \text{and} \quad K_0 \equiv 1.$$

The solution u to this “no enemy observers” problem is shown in Figure 16C. Figure 16A shows the contours of f_0 with two locally optimal trajectories. The ‘upper’ solid trajectory is globally optimal and found by tracing the gradient of u ; the ‘lower’ locally optimal trajectory is computed using PMP.

Our perception of the trajectory safety will change once we discover specific locations of enemy observers. For example, if we know that there are two observers located at $\mathbf{x}_1 = (0.50, 0.77)$ and $\mathbf{x}_2 = (0.33, 0.45)$, we might encode this new information in the cost function:

$$K(\mathbf{x}) = 1 + 2 \exp\left(\frac{|\mathbf{x} - \mathbf{x}_1|^2}{0.01}\right) + 8 \exp\left(\frac{|\mathbf{x} - \mathbf{x}_2|^2}{0.002}\right).$$

The solution to (4.3) with speed f_0 and the above cost K can be shown to satisfy (1.1) with $f = f_0/K$. The contours of this new modified speed f can be seen in Figure 16B with two “locally

safest” trajectories that can be viewed as perturbations of the locally time-optimal paths from Figure 16A. Note that, because of the higher cost around \mathbf{x}_1 , the ‘upper’ locally optimal trajectory is no longer globally optimal.

The full solution to the time-optimal problem becomes useful if we want to introduce A* techniques for all “multiple enemy observers” problems. Let $V_0(\mathbf{x})$ be the minimum time to reach \mathbf{s} using the speed f_0 . Suppose that V_0 is pre-computed by FMM and stored for the entire X . Returning to the problem with known observers, we may take $\varphi = V_0$ since $f_0 \leq f \implies V_0 \leq V$. The overestimate Ψ can be obtained by integrating K/f_0 along any feasible trajectory. If we use the globally time-optimal trajectory (the solid black curve in Figure 16A), this yields a good $\Psi_A \approx 0.6752$. An even better overestimate $\Psi_B \approx 0.6447$ is obtained if we use the globally time-optimal trajectory as the initial guess for PMP, and then integrate K/f_0 along the resulting “locally safest” trajectory (the ‘upper’ black curve in Figure 16B).

Figure 16D shows the solution level sets for the “multiple enemy observers” problem together with boundaries of several computational sets. The bold black curve is ∂L , showing the part of $\bar{\Omega}$ **ACCEPTED** by FMM. The next (inward) bold curve is ∂C_2 with $\Psi = \Psi_B$ – the boundary of a subset **ACCEPTED** by AA*-FMM. The final bold curve is ∂C_2 with $\Psi = U(\mathbf{s})$ – this approximates the boundary of a subset **ACCEPTED** by SA*-FMM. (AA*-FMM would also restrict to the latter set, but only if we were lucky enough to start with Ψ corresponding to the ‘lower’ curve in Figure 16B).

Even though AA*-FMM computes the solution on a larger part of the domain, its computational efficiency is still comparable and the accuracy is superior to SA*-FMM. For example, with $m = 201$ (using 100 trial runs averaged for the time) we have

Method	Time (seconds)	Ratio \mathcal{P}	Error \mathcal{E}_N^*
<i>FMM</i>	0.02665	0.82	0
<i>SA*-FMM</i>	0.004950	0.130	0.02550
<i>AA*-FMM</i> , $\Psi = \Psi_B$	0.0101	0.29	1.77×10^{-11}
<i>AA*-FMM</i> , $\Psi = U(\mathbf{s})$	0.00526	0.14	0.00150

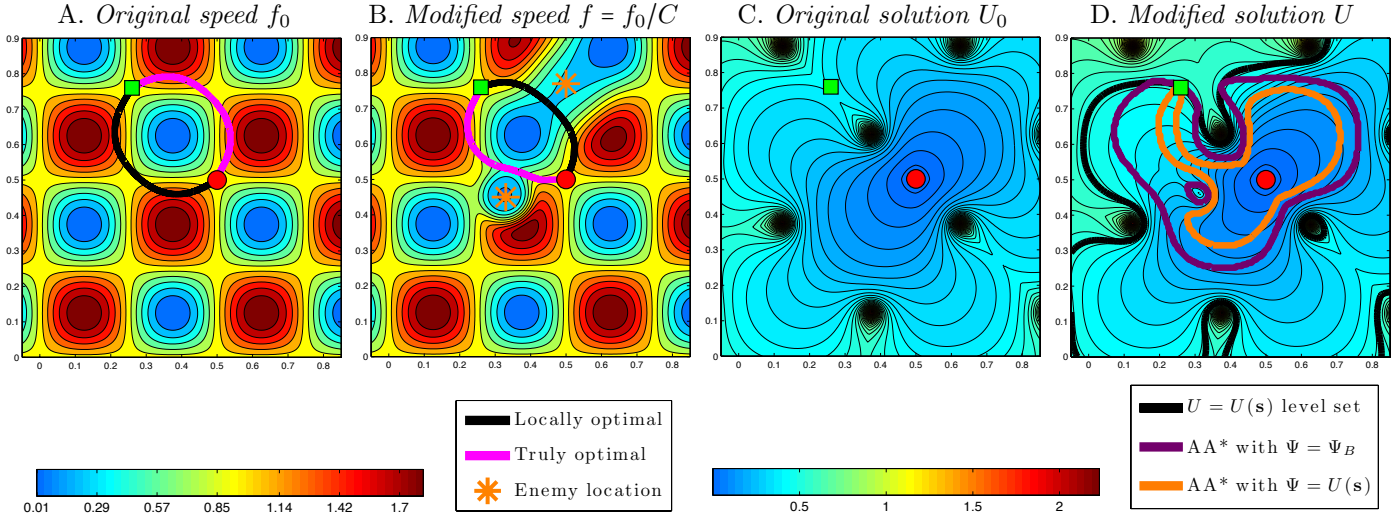


Figure 16: **A.** Contours of the original speed function f_0 . **B.** Contours of “modified speed function” $f = f_0/K$ with the enemy locations shown by asterisks. **C.** Contours of the original solution to the problem with speed f_0 and constant running cost. **D.** Contours of the solution corresponding to the modified speed function $f = f_0/K$ with ∂L drawn in bold black. ∂C_2 is in dark purple using $\Psi = \Psi_B$, and in orange when using $\Psi = U(\mathbf{s})$.

5 CONCLUSIONS

We have described a new A*-type modification of the Fast Marching Method solving Eikonal equations for a ‘single source/target’ problem. Unlike the prior methods for this problem, which were developed to mirror in the ‘standard A*’ algorithm on graphs [13], our approach is based on a lesser known ‘alternative A*’ [5]. These prior SA*-FMM methods [11, 15, 16, 17, 18, 29, 30] either introduce additional errors that vanish slowly (if at all) under grid refinement, or must accept a much larger portion of the domain. In contrast, our AA*-FMM is able to significantly restrict computations, with any additional errors quickly decreasing under grid refinement.

One weakness of AA*-FMM is the reliance on an overestimate Ψ , especially when the feasibility of any (\mathbf{s}, \mathbf{t}) trajectory is in question (e.g., in the presence of obstacles). A good Ψ can be also found from related control problems or based on Pontryagin Maximum Principle (PMP). Here we mention two more approaches not tested in the current paper:

- One can use $\Psi = \zeta U^C(\mathbf{s})$, where $\zeta > 1$ and U^C is the solution found by FMM on a much coarser grid.
- One can also use the output of SA*-FMM on the same grid with an aggressive/inconsistent φ , setting $\Psi = U^*(\mathbf{s})$.

In the latter case, AA*-FMM should be viewed as a post-processing technique to improve the accuracy. This might seem superfluous: after all, PMP could also be applied using the output of SA*-FMM as an initial guess. But as we show in Figures 10 and 15, the errors from SA*-FMM are likely to result in PMP converging to some other (locally, rather than globally) optimal trajectory.

The effectiveness of the AA* domain restriction depends on the *quality* of φ and Ψ . If the initial Ψ is overly conservative, it can also be improved dynamically using the Branch & Bound techniques. No benchmarking results for the latter approach were included here for the sake of brevity.

We also list several desirable future extensions with significant impact on applications. First, AA* can be used instead of SA* within D* and E* path replanners [11, 19]. Second, the original AA* on graphs is applicable in both label-setting and label-correcting algorithms. It should not be hard to incorporate the same idea into other non-iterative and fast iterative methods for Hamilton-Jacobi PDEs. Our preliminary results for the Locking Sweeping Method [1] prove the feasibility of this approach. Third, since many gridpoints will never be used, allocating memory for the entire grid may be wasteful (particularly in high dimensions). One approach, described in [16, 17, 18], is to allocate gridpoints as needed and make use of a hash lookup table. Our current implementation of AA*-FMM does not use this idea, but we hope to explore it in the future. Finally, we note that all of the A* techniques can be also trivially extended to problems with a single-source and multiple targets. Similar underestimates can be also built for a moderately large set of sources $\{\mathbf{s}_i\}$ (e.g., $\varphi = \min_i \varphi_i^0$).

The error analysis in the Appendix relies on a conjecture, which so far has been only proven for a linear advection equation. For the Eikonal case, we currently rely on experimental/numerical confirmation. Nevertheless, we believe that a similar approach will be also useful in analyzing errors in more general domain restriction problems; e.g., for the errors due to an ‘almost causal’ domain decomposition in [6].

Acknowledgements. The authors would like to thank Slav Kirov for his contributions to the initial part of this project during the 2010-REU at Cornell University. We would also like to thank Gabriel Peyré for his input on the satellite image example used in Section 4.3.

6 APPENDIX: WHY DOES IT CONVERGE?

If AA*-FMM is used with an inconsistent heuristic φ , the domain restriction usually affects the dependency graph (i.e., $G(\mathbf{s}) \notin \hat{X}$), and the produced solution is larger than would result from

running FMM on the full grid: $U^*(\mathbf{s}) > U(\mathbf{s})$. In this section we analyze why $(U^*(\mathbf{s}) - U(\mathbf{s})) \rightarrow 0$ as $h \rightarrow 0$.

We first note that the answer is simple if there exists an open set $\Omega_0 \subset \Omega$ such that

- the (\mathbf{s}, \mathbf{t}) -optimal trajectory lies in Ω_0 , and
- and all gridpoints falling into Ω_0 are accepted by AA*-FMM *regardless of* h .

In this case, an $\bar{\Omega}_0$ -constrained viscosity solution will already yield the correct $u(\mathbf{s})$ in the limit. In previous sections, we showed that such Ω_0 often arises because Ψ and/or φ are not tight. But if the over/underestimates also improve in quality as $h \rightarrow 0$, then the AA*-FMM accepted region shrinks under grid refinement, and a more careful argument is needed to explain the convergence.

To address this, we compare solutions produced by the original FMM solving the same discretized system (3.2) but on different grid subsets and with different boundary conditions. For the rest of this section, we will not rely on the fact that \hat{X} is defined through AA*-FMM. As a benefit, our error analysis is also relevant for domain decomposition-based parallelizations of FMM; e.g., see [6].

Consider a restriction of FMM computations to any $\hat{X} \subset X$ containing both \mathbf{s} and \mathbf{t} , and define the “restriction boundary” set $\Xi = \{\mathbf{x} \in X \setminus \hat{X} \mid N(\mathbf{x}) \cap \hat{X} \neq \emptyset\}$. For notational simplicity, we will assume that the (\mathbf{s}, \mathbf{t}) -optimal trajectory is unique and that the upwind neighbors $(\mathbf{x}_H, \mathbf{x}_V)$ are uniquely defined for every gridpoint \mathbf{x} .

We will discuss the relationship between the following discretized solutions:

- As before, U denotes the solution on the entire X with the boundary condition $U(\mathbf{t}) = 0$.
- \hat{U} denotes the solution on \hat{X} with the same boundary condition $\hat{U}(\mathbf{t}) = 0$. We can also interpret it as a solution on $\hat{X} \cup \Xi$ with $Q = \{\mathbf{t}\} \cup \Xi$ and $q = +\infty$ on Ξ . Recall that, if \hat{X} is defined as the set of nodes **ACCEPTED** by AA*-FMM, then this method also produces the same solution (i.e., $U^* = \hat{U}$ on \hat{X}).
- \bar{U} denotes the solution computed on $\hat{X} \cup \Xi$ with $\bar{U}(\mathbf{t}) = 0$ and the more general boundary conditions $\bar{U}(\mathbf{x}_i) = q_i$ specified $\forall \mathbf{x}_i \in \Xi$.

Observation 6.1. The following properties are easy to verify based on the causality of (3.2):

1. $q_i = U_i, \forall \mathbf{x}_i \in \Xi \implies \bar{U}_j = U_j, \forall \mathbf{x}_j \in \hat{X}$;
2. $q_i \geq U_i, \forall \mathbf{x}_i \in \Xi \implies \bar{U}_j \geq U_j, \forall \mathbf{x}_j \in \hat{X}$;
3. $\hat{U}_j \geq \bar{U}_j, \forall \mathbf{x}_j \in \hat{X}$;
4. Suppose C is a constant such that $C \geq \max_{\mathbf{x}_j \in \hat{X}} \hat{U}_j$. Then $q_i \geq C, \forall \mathbf{x}_i \in \Xi \implies \bar{U}_j = \hat{U}_j, \forall \mathbf{x}_j \in \hat{X}$.
5. Suppose $D(\mathbf{x})$ is the arclength of the shortest grid-aligned path within \hat{X} from \mathbf{x} to \mathbf{t} . Then $C = \max_{\mathbf{x}_j \in \hat{X}} D(\mathbf{x})/F_1 \geq \max_{\mathbf{x}_j \in \hat{X}} \hat{U}_j$.

For any specific $\mathbf{x}_i \in X$, if we define $\hat{X} = X \setminus \{\mathbf{x}_i\}$ and choose $q_i > U_i$ this might result in $\hat{U}(\mathbf{s}) > U(\mathbf{s})$. This “add-one-gridpoint-to- Q ” procedure motivates our definition of *sensitivity coefficients*:

$$\alpha_i = \alpha(\mathbf{x}_i) = \frac{\partial U(\mathbf{s})}{\partial U_i} \text{ or, more rigorously, } \alpha_i = \frac{\partial \hat{U}(\mathbf{s})}{\partial q_i} \text{ computed on } \hat{X} = X \setminus \{\mathbf{x}_i\} \text{ with } q_i = U_i.$$

Due to the monotonicity of (3.2), $\alpha_i \geq 0$ and it is strictly positive if and only if $\mathbf{x}_i \in G(\mathbf{s})$.

Lemma 6.2. *The net effect of a domain restriction can be bounded from above using α ’s even for a general set \hat{X} :*

1. If $q(\mathbf{x}) \geq U(\mathbf{x}), \forall \mathbf{x} \in \Xi$, then $\bar{U}(\mathbf{s}) - U(\mathbf{s}) \leq \sum_{\mathbf{x} \in \Xi} \alpha(\mathbf{x}) (q(\mathbf{x}) - U(\mathbf{x}))$.

2. If $C \geq \hat{U}(\mathbf{x})$, $\forall \mathbf{x} \in \hat{X}$, then $\hat{U}(\mathbf{s}) - U(\mathbf{s}) \leq C \sum_{\mathbf{x} \in \Xi} \alpha(\mathbf{x})$.

Proof. The upwind finite difference discretization (3.3) is equivalent to a semi-Lagrangian discretization:

$$U(\mathbf{x}_{ij}) = \min_{\beta \in [0,1]} \left\{ \frac{|\beta \mathbf{x}_H + (1-\beta) \mathbf{x}_V - \mathbf{x}_{ij}|}{f(\mathbf{x})} + \beta U(\mathbf{x}_H) + (1-\beta) U(\mathbf{x}_V) \right\}. \quad (6.1)$$

Despite the very different Eulerian perspective and notation, (3.3) can be actually derived from Kuhn-Tucker optimality conditions for (6.1); see [27, 26, 28]. Moreover, the latter can be also viewed as the dynamic programming equation for a *Stochastic Shortest Path Problem* on the grid X ; see [28] for a detailed discussion. In this interpretation, the transition from \mathbf{x}_{ij} to the neighboring node (either \mathbf{x}_H or \mathbf{x}_V) happens probabilistically, with respective probabilities β and $(1-\beta)$, and $\frac{|\beta \mathbf{x}_H + (1-\beta) \mathbf{x}_V - \mathbf{x}_{ij}|}{f(\mathbf{x})}$ is the cost we incur for choosing this probability distribution. The process continues until we reach \mathbf{t} , and the goal is to select $\beta_* : X \rightarrow [0,1]$ that minimizes the expected cumulative cost up to that termination. We note that, for $\mathbf{x}_{ij} = \mathbf{s}$, we have $\alpha(\mathbf{x}_V) = (1 - \beta_*(\mathbf{s}))$, $\alpha(\mathbf{x}_H) = \beta_*(\mathbf{s})$ and α values on the rest of $G(\mathbf{s})$ can be similarly computed using (6.1) recursively; see [7]. Moreover, if we start from \mathbf{s} and use the optimal “stochastic routing policy” $\beta_*(\cdot)$, then $\alpha(\mathbf{x})$ can be naturally interpreted as a probability of passing through \mathbf{x} before arriving at \mathbf{t} .

Suppose now we use $\beta_*(\cdot)$, but on a \hat{X} -restricted problem, starting from \mathbf{s} and terminating the process (+ paying the additional cost of $q(\mathbf{x})$) if we transition into any $\mathbf{x} \in \Xi$ before reaching \mathbf{t} . Denote by \tilde{U} the expected total cost of using this policy and by $\tilde{\alpha}(\mathbf{x})$ the probability of reaching \mathbf{x} before termination. We first note that $\tilde{\alpha}(\mathbf{x}) \leq \alpha(\mathbf{x})$, $\forall \mathbf{x} \in \hat{X} \cup \Xi$ since some stochastic paths previously leading through \mathbf{x} are now removed due to an earlier entry to Ξ . Secondly, $\tilde{U} \geq \bar{U}$, since the latter is found by optimizing over all possible $\beta : \hat{X} \rightarrow [0,1]$, including the restriction of $\beta_*(\cdot)$. Thus,

$$\bar{U}(\mathbf{s}) - U(\mathbf{s}) \leq \tilde{U}(\mathbf{s}) - U(\mathbf{s}) = \sum_{\mathbf{x} \in \Xi} \tilde{\alpha}(\mathbf{x}) (q(\mathbf{x}) - U(\mathbf{x})) \leq \sum_{\mathbf{x} \in \Xi} \alpha(\mathbf{x}) (q(\mathbf{x}) - U(\mathbf{x})),$$

which completes the proof of part 1. To prove part 2, select $q(\mathbf{x}) = C$, $\forall \mathbf{x} \in \Xi$. Since the exit-penalty C is prohibitively high, the stochastic path starting from $\mathbf{s} \in \hat{X}$ and using the optimal routing policy will avoid Ξ with probability 1. Thus, $\bar{U}(\mathbf{s}) = \tilde{U}(\mathbf{s})$ (see the last part of Observation 6.1), and using the above result

$$\hat{U}(\mathbf{s}) - U(\mathbf{s}) \leq \sum_{\mathbf{x} \in \Xi} \alpha(\mathbf{x}) (C - U(\mathbf{x})) \leq C \sum_{\mathbf{x} \in \Xi} \alpha(\mathbf{x}). \quad \square$$

Let $d(\mathbf{x})$ be the distance from \mathbf{x} to the characteristic passing through \mathbf{s} (i.e., the (\mathbf{s}, \mathbf{t}) -optimal trajectory).

Conjecture 6.3. There exists a constant $\rho > 0$ such that, for small enough h , $\alpha(\mathbf{x}) \leq e^{-\rho[d(\mathbf{x})]^2/h}$.

As of right now, we only have a rigorous proof of this statement for an upwind discretization of a constant-coefficient advection PDE [7, Chapter 4]. The same proof also covers the Eikonal equation when all characteristics are parallel, but this clearly does not hold for the case $Q = \{\mathbf{t}\}$. Still, the numerical evidence (see Figure 17) indicates that this exponential decay is also present in the current context as well.

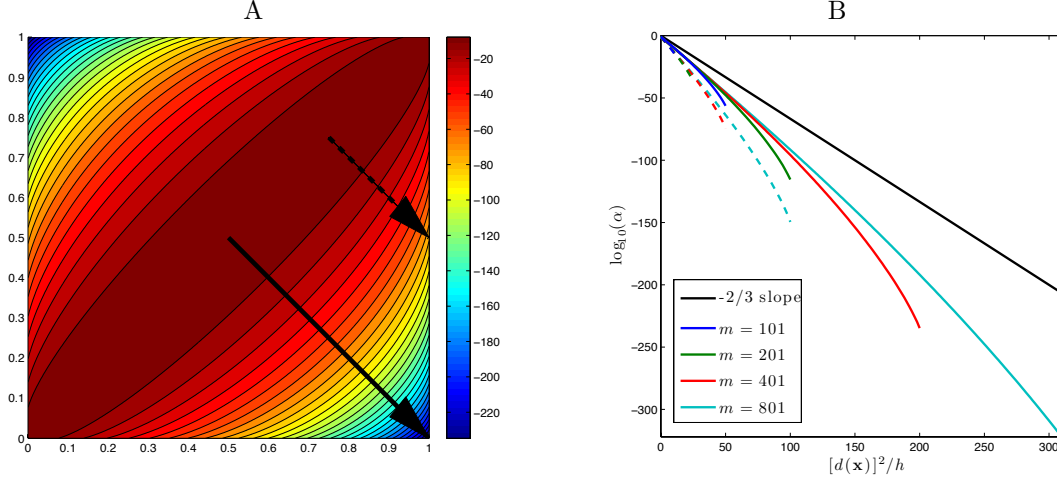


Figure 17: Alpha values decaying away from the characteristic. Subfigure **A**: shows the level sets of $\log_{10}(\alpha)$ for the constant speed example considered in §4.1. The solid and dashed arrows are perpendicular to the (\mathbf{s}, \mathbf{t}) -optimal trajectory. Subfigure **B** shows the rate of decay of $\log_{10}(\alpha)$ along each of these arrows computed for several different grid resolutions.

Theorem 6.4. *Let $\{X^h\}$ be a family of Cartesian grids on Ω with gridsize $h = 1/(m-1)$ such that both \mathbf{s} and \mathbf{t} are gridpoints for all m . Define $\hat{X}^h = \{\mathbf{x} \in X^h \mid d(\mathbf{x}) < r\}$, where $r = O(h^\mu)$, for some $\mu \in [0, \frac{1}{2})$. Let U^h and \hat{U}^h be numerical solutions of the system (3.2) on X^h and \hat{X}^h respectively. If Conjecture 6.3 holds, then $(\hat{U}^h(\mathbf{s}) - U^h(\mathbf{s})) \rightarrow 0$ as $h \rightarrow 0$.*

Proof. We note that the n -volume of the optimal-trajectory-centered r -cylinder approaches zero, though the total number of gridpoints in \hat{X}^h grows as $h \rightarrow 0$. For convenience, we also define $k = r^2/h = O(h^{2\mu-1})$, which tends to $+\infty$ as $h \rightarrow 0$. If S is the path length of the (\mathbf{s}, \mathbf{t}) -optimal trajectory, then the number of gridpoints in Ξ^h is $O(\frac{Sr^{n-2}}{h^{n-1}}) = O(k^\nu)$, where $\nu = \frac{(n-1)-\mu(n-2)}{1-2\mu} > 0$. Considering the shortest grid-aligned and \hat{X}^h -constrained path from any $\mathbf{x} \in \hat{X}^h$ to \mathbf{t} , it is easy to show that $D^h = (S+r)\sqrt{n}$ is the upper bound for that path's length. Thus, $C^h = D^h/F_1$ is an upper bound for $\max_{\mathbf{x} \in \hat{X}^h} \hat{U}^h(\mathbf{x})$. If Conjecture 6.3 holds, then asymptotically $\alpha^h(\mathbf{x}) \leq e^{-\rho r^2/h} = e^{-\rho k}$, for all $\mathbf{x} \in \Xi_h$. By Lemma 6.2, $(\hat{U}^h(\mathbf{s}) - U^h(\mathbf{s}))$ is bound from above by $\left[C^h \sum_{\mathbf{x} \in \Xi^h} \alpha^h(\mathbf{x}) \right] = O(k^\nu e^{-\rho k})$, which converges to 0 under grid refinement. \square

REFERENCES

- [1] S. Bak, J. McLaughlin, & D. Renzi, *Some Improvements for the Fast Sweeping Method*, SIAM J. Sci. Comput., Vol. 32, No. 5, pp. 2853-2874, 2010.
- [2] M. Bardi and I. Capuzzo-Dolcetta, *Optimal Control and Viscosity Solutions of Hamilton Jacobi-Bellman Equations*, Birkhäuser, 1997.
- [3] G. Barles and P. E. Souganidis, *Convergence of approximation schemes for fully nonlinear second order equations*, Asymptot. Anal., 4:271-283, 1991.
- [4] R.E. Bellman, *Dynamic Programming*, Princeton University Press, Princeton, NJ, 1957.
- [5] D.P. Bertsekas, *Dynamic Programming and Optimal Control*, 2nd Edition, Volumes I and II, Athena Scientific, Boston, MA, 2001.
- [6] Cacace, S., Cristiani, E., Falcone, M., Picarelli, A. *A patchy Dynamic Programming scheme for a class of Hamilton-Jacobi-Bellman equations*, SIAM J. Sci. Comp. Vol. 34, No. 5, pp. A2625-A2649, 2012.
- [7] A. Chacon, *Eikonal Equations: new two-scale algorithms and error analysis*, Ph.D. Thesis, Cornell University, 2013.

- [8] A. Chacon & A. Vladimírsky, *Fast two-scale methods for Eikonal equations*, SIAM J. on Scientific Computing, 34/2, 2012.
- [9] M.G. Crandall, P.-L. Lions, *Viscosity solutions of Hamilton-Jacobi equations*, Transactions of the American Mathematical Society, 277 (1), pp. 1–42, 1983.
- [10] E.W. Dijkstra, *A Note on Two Problems in Connexion with Graphs*, Numerische Mathematik, Vol. 1, pp. 269–271, 1959.
- [11] D. Ferguson & A. Stentz, *Field D*: An interpolation-based path planner and replanner*, Proceedings of International Symposium on Robotics Research (ISRR), 2005.
- [12] A.V. Goldberg & C. Harrelson, *Computing the Shortest Path: A* Search Meets Graph Theory*, Technical Report, Microsoft Research, 2004.
- [13] P.E. Hart, N.J. Nilsson, & B. Raphael, *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*, IEEE Transactions of Systems Science and Cybernetics, Vol. SSC-4, No. 2, pp. 100–107, 1968.
- [14] Kimmel, R. & Sethian, J.A., *Fast Marching Methods on Triangulated Domains*, Proc. Nat. Acad. Sci., 95, pp. 8341–8435, 1998.
- [15] C. Pètrès, *Trajectory Planning for Autonomous Underwater Vehicles*, Heriot-Watt University, PhD Dissertation, 2007.
- [16] G. Peyré & L.D. Cohen, *Heuristically Driven Front Propagation for Geodesic Paths Extraction*, Proc. of VLISM '05 (N. Paragios, O. D. Faugeras, T. Chan, C. Schnörr, eds.), Springer, vol. 3752, pp. 173–185, 2005.
- [17] G. Peyré & L.D. Cohen, *Landmark-Based Geodesic Computation for Heuristically Driven Path Planning*, Proc. of CVPR '06, IEEE Computer Society, pp. 2229–2236, 2006.
- [18] G. Peyré & L.D. Cohen, *Heuristically Driven Front Propagation for Fast Geodesic Path Extraction*, International Journal for Computational Vision and Biometrics Vol. 1, No. 1, pp. 55–67, 2008.
- [19] R. Philippsen, *A Light Formulation of the E* Interpolated Path Replanner*, Technical report, Autonomous Systems Lab, École Polytechnique Fédérale de Lausanne, 2006.
- [20] I. Pohl, *Bi-directional Search*, Machine Intelligence, vol. 6, eds. Meltzer and Michie, Edinburgh University Press, pp. 127–140, 1971.
- [21] L. S. Pontryagin, V. Boltyanskii, R. V. Gamkrelidze, & E. F. Mishenko, *The Mathematical Theory of Optimal Processes*, Wiley, 1962.
- [22] E. Rouy & A. Tourin, *A Viscosity Solutions Approach to Shape-From-Shading*, SIAM J. Num. Anal., 29, 3, pp. 867–884, 1992.
- [23] J.A. Sethian, *A Fast Marching Level Set Method for Monotonically Advancing Fronts*, Proc. Nat. Acad. Sci., 93, 4, pp. 1591–1595, February 1996.
- [24] J.A. Sethian, *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision and Materials Sciences*, Cambridge University Press, 1996.
- [25] J.A. Sethian & A. Vladimírsky, *Fast Methods for the Eikonal and Related Hamilton–Jacobi Equations on Unstructured Meshes*, Proc. Nat. Acad. Sci., 97, 11 (2000), pp. 5699–5703.
- [26] J.A. Sethian & A. Vladimírsky, *Ordered Upwind Methods for Static Hamilton-Jacobi Equations: Theory & Algorithms*, SIAM J. on Numerical Analysis 41, 1, pp. 325–363, 2003.
- [27] J.N. Tsitsiklis *Efficient Algorithms for Globally Optimal Trajectories*, IEEE Tran. Automatic Control, 40, pp. 1528–1538, 1995.
- [28] A. Vladimírsky, *Label-setting methods for Multimode Stochastic Shortest Path problems on graphs*, Mathematics of Operations Research 33(4), pp. 821–838, 2008.
- [29] D.S. Yershov, S.M. LaValle, *Simplicial Dijkstra and A* Algorithms for Optimal Feedback Planning*, in Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2011.
- [30] D.S. Yershov, S.M. LaValle, *Simplicial Dijkstra and A* Algorithms: From Graphs to Continuous Spaces*, Advanced Robotics, Vol. 26, no. 17, pp. 2065–2085, 2012.
- [31] H. Zhao, *A Fast Sweeping Method for Eikonal Equations*, Mathematics of Computation, Vol. 74, Num. 250, pp. 603–627, 2004.