# Simplicial Dijkstra and A* Algorithms for Optimal Feedback Planning

Dmitry S. Yershov and Steven M. LaValle

***Abstract*— This paper considers the Euclidean shortest path problem among obstacles in $\mathbb{R}^n$. Adaptations of Dijkstra's and A* algorithms are introduced that compute the approximate cost-to-go function over a simplicial complex embedded in the free space. Interpolation methods are carefully designed and analyzed so that they are proven to converge numerically to the optimal cost-to-go function. As the result, the computed function produces approximately optimal trajectories. The methods are implemented and demonstrated on 2D and 3D examples. As expected, the simplicial A* algorithm significantly improves performance over the simplicial Dijkstra's algorithm.**

***Index Terms*— Optimal control, feedback motion planning, Bellman's principle, shortest paths, Dijkstra's algorithm, A* algorithm**

## I. INTRODUCTION

Computing the Euclidean shortest path to a given goal is a recurring problem in robotics. In addition to optimal robot navigation and manipulation, it is also useful in image processing, financial modeling, physics, etc. We focus on finding the shortest path between a given point and a polygonal goal set in an $n$-dimensional environment with polygonal obstacles. For $n = 3$ this problem is already PSPACE-hard [3]; therefore, approximation methods have been developed [4], [17], [21]. We, therefore, consider only approximate shortest paths.

In robotics, algorithms that compute approximately optimal paths are based on a common approach to discretizing the problem: construct a reachability graph for a robot in a given environment using, for example, regular grids [14], [18]; and apply a graph search algorithm to find the shortest path. Although this approach is appealingly intuitive, the computed paths do not necessarily converge to the shortest path as the graph resolution increases.

An improved approach is the continuous optimal cost-to-go formulation of the shortest path problem [5], [7]. The optimal feedback law (solution to the corresponding optimal control problem) gives the shortest path when integrated. In this case, the space of possible feedback laws is discretized instead of the path space. This provides flexibility in choosing the optimal path from a continuum of all possible paths. Moreover, this approach does not require an implementation of path following controllers; the motion strategy is instead naturally given by the feedback law. In this sense, this approach can be considered as an optimal version of a navigation function [19].

D.S. Yershov and S.M. LaValle are with the Department of Computer Science at University of Illinois at Urbana-Champaign, Urbana, Illinois, 61801, USA {yershov2,lavalle}@illinois.edu

Algorithms proposed in this paper extend the control-theoretic approach outlined above. The main advantages of the new method are:

- The feedback law is defined through interpolation, thereby providing control values at every point in space, not just at a discrete set of points. Moreover, the interpolation technique is introduced for spaces of any dimension and a general simplicial decomposition of the environment, extending existing interpolation techniques beyond regular grids in 2D or 3D [21] and 2D triangulations of manifolds [13].
- The interpolation scheme inherits the causality property of the original problem. We exploit this property to build a Dijkstra-like [6] algorithm to solve the resulting system of nonlinear equations in one sweep through the domain. This provides an extremely efficient algorithm for computing a feedback plan with an asymptotic running time $\mathcal{O}(N \log N)$ (here $N$ is the number of vertices in the simplicial decomposition).
- In case the initial point is known, we further reduce the computational cost by proposing a continuous version of the A* algorithm [10]. Introducing the heuristic into the interpolation-based algorithm is nontrivial, and care must be taken to ensure that the system of discrete dynamic programming equations is solved correctly.
- The theoretical framework sketched in this paper provides error bounds for the proposed algorithms. This analysis paves the way for showing convergence of the approximate path to the optimal path as the resolution of the simplicial mesh is refined.

The proposed approach concerns computations only, and it closely resembles the numerical analysis framework for the Eikonal equation in [20]. In this respect, our method is different from [7], [14], [18], which address the problem of simultaneous plan computation, path execution and dynamic replanning. However, the approach can be thought of as a planner that, if embedded into simultaneous execution and replanning framework, possibly leads to an even more general interpolation-based methodology.

## II. PROBLEM FORMULATION

Consider the problem of optimal feedback planning in a $n$-dimensional Euclidean space. High dimensional environments may arise from considering the configuration space of a robot. Assume that the robot's coordinates are restricted by global constraints only. Hence, the robot's coordinates $x$ are free to translate in $X_{\text{free}} = \mathbb{R}^n \setminus X_{\text{obs}}$, in which $X_{\text{obs}}$ is an open set with $(n-1)$-dimensional polygonal boundary. Finally, assume the goal set, $X_{\text{goal}} \subset X_{\text{free}}$, is a closed set

with polygonal boundary. The problem is to find a feedback plan that navigates the robot along the shortest path from some initial state, $x_{\mathrm{I}} \in X_{\mathrm{free}}$, to the goal set, while avoiding obstacles.

Formally the optimal feedback plan can be described as a vector valued function $F(x)$, such that the solution to the initial value problem

$$\dot{x}(t) = F(x(t)), \quad x(0) = x_{\mathrm{I}} \tag{1}$$

is the shortest path. Generally, we may consider discontinuous feedback plans, and hence the Filippov solution in (1) is assumed [8].

It is well known that the optimal feedback plan can be derived from the *cost-to-go* function, $V(x)$, which satisfies Bellman's equation [2]:

$$V(x) = \liminf_{\delta \to 0} \inf_{h \in B(\delta)} \left\{ V(x+h) + \|h\| \right\}, \tag{2}$$

in which $B(\delta)$ is a ball of radius $\delta$ centered at the origin. Once $V(x)$ is known, $F(x) = -\nabla V(x)$ is the optimal feedback plan.

To summarize, the main goal of this paper is to compute the cost-to-go function, and use the result to derive the optimal feedback plan.

## III. NUMERICAL APPROACH

Equation (2) admits an analytical solution in special cases. For example, if there are no obstacles in the environment, $V(x)$ is simply given by the distance function to $X_{\mathrm{goal}}$. For a two-dimensional environment with polygonal obstacles, this problem can be solved exactly by visibility graph methods [12] or continuous Dijkstra [11], [15], [16]. However, the exact solution is not known under general conditions, and thus we must rely on a numerical approximation, which we discuss in this section.

### A. Approximating $V(x)$

We construct a simplicial discretization of $X_{\mathrm{free}}$ by choosing a set of vertices $X_{\mathrm{d}} = \{x_i \in X_{\mathrm{free}} \mid 1 \leq i \leq N\}$ (a subset of $X_{\mathrm{free}}$). Further, define an abstract simplicial complex, $\mathcal{T} = \{T \subseteq \{1, \ldots, N\}\}$, such that if $T' \subseteq T \in \mathcal{T}$, then $T' \in \mathcal{T}$. In this case, $T'$ is called a *face* of $T$; if additionally $T' \neq T$, then it is called a *proper face* of $T$. In this regard, the notion of (proper) faces is parallel to the notion of (proper) subsets. Next, denote a geometric representation of simplex $T \in \mathcal{T}$ as $X(T)$ such that $X(T)$ is the convex hull of the set $\{x_i\}_{i \in T}$. The tuple $(X_{\mathrm{d}}, \mathcal{T})$ is called a simplicial complex if any two simplices intersect over the common proper face only, i.e., for any $T$ and $T'$ in $\mathcal{T}$, $X(T) \cap X(T') = X(T \cap T')$. Finally, a simplicial complex discretizes $X_{\mathrm{free}}$, if $\bigcup_{T \in \mathcal{T}} X(T) = X_{\mathrm{free}}$.[1]

Next, build a piecewise linear approximation $\hat{V}$ of the cost-to-go function, using a simplicial discretization of $X_{\mathrm{free}}$. Let the approximation take value $\hat{V}_i$ at vertex $x_i$. Define $\hat{V}(x)$ by linear interpolation within simplex $T \in \mathcal{T}$, the geometric description of which contains $x$:

$$\hat{V}(x) = \hat{V}\Big( \sum_{i \in T} \alpha_i x_i \Big) \triangleq \sum_{i \in T} \alpha_i \hat{V}_i, \tag{3}$$

in which $\alpha_i \geq 0$ for all $i$, and $\sum_{i \in T} \alpha_i = 1$. The values $\alpha_i$ are called *barycentric coordinates* of $x$ within $X(T)$. The approximation is completely determined by its values at the vertices of a simplicial complex through (3).

### B. Discrete dynamic programming

Since a piecewise linear function cannot satisfy (2) under general conditions, we introduce a discrete version of Bellman's principle by considering (2) at points of $X_{\mathrm{d}}$ only:

$$\hat{V}(x_i) = \min_{T \in \mathcal{N}(i)} \inf_{x \in X(T_i)} \left\{ \hat{V}(x) + \big\| x_i - x \big\| \right\}, \tag{4}$$

in which $\mathcal{N}(i) = \{T \in \mathcal{T} \mid i \in T\} \subset \mathcal{T}$ is a set of simplices incident to vertex $x_i$, $T_i = T \setminus \{i\}$ is a proper face of simplex $T$ opposite vertex $x_i$.

To construct a fully discrete numerical method we closely follow [20] by using linear interpolation (3) to solve the minimization problem (4) at each vertex. A similar discretization for the Hamilton-Jacobi equation is introduced in [1] based on upwind differencing, whereas our approach is based directly on discretization of the dynamic programming principle (firstly introduced in [21]), and it generalizes to high-dimensional simplicial grids.

## IV. ALGORITHMIC APPROACH

The discrete dynamic programming principle (4), considered at all vertices of the discretization, describes a system of nonlinear equations. An application of standard iterative nonlinear solvers suffers from several shortcomings: it requires a sufficiently accurate initial guess, running time is high, and the result is only an approximate solution. By contrast, discrete graph search methods, such as Dijkstra's algorithm [6] or A* algorithm [10], solve a similar system of dynamic programming equations in optimal time without requiring an initial guess. In this paper we implement a modification of these algorithms to solve the given system.

### A. Simplicial Dijkstra algorithm

We propose a Simplicial Dijkstra algorithm (SDA) that evaluates the function $\hat{V}$ in increasing order of its values using a priority queue, similarly to Dijkstra's graph search algorithm. Under minimal conditions, our implementation guarantees that equation (4) is solved only once for each vertex. Thus, the entire computation is done in one "sweep" through the simplicial complex; see Algorithm 1 for details.

Algorithm 1 is identical to Dijkstra's graph search algorithm if the complex is a graph (i.e, a 1-complex), and **minloc** is replaced with the minimum cost over all paths to neighboring vertices. In our case, however, **minloc** is defined to satisfy (4) for general simplicial complexes.

[1] If the boundary of $X_{\mathrm{free}}$ is not a polygonal set, then a simplicial discretization may not exist. Although, it is still possible to find a simplicial complex such that $\bigcup_{T \in \mathcal{T}} X(T) \subset X_{\mathrm{free}}$, and the difference $X_{\mathrm{free}} \setminus \bigcup_{T \in \mathcal{T}} X(T)$ is "small". In this case the shortest path in $X_{\mathrm{free}}$ can be approximated by the shortest path in the simplicial complex.

**Algorithm 1** Simplicial Dijkstra

**Input:** Simplicial complex $(X_\mathrm{d}, \mathcal{T})$, goal set $X_\mathrm{goal}$
**Output:** Approximation of cost-to-go function, $\hat{V}_i$, at all vertices $x_i$ of simplicial complex

Initialize priority queue $Q$ of all vertex indices. Set priority key $\hat{K}_i \leftarrow 0$, for all $x_i \in X_\mathrm{goal}$, and $\hat{K}_i \leftarrow \infty$, otherwise
**while** $Q$ is not empty **do**
  Pop $j$ with least key $\hat{K}_j$ from $Q$
  Set $\hat{V}_j \leftarrow \hat{K}_j$
  **for all** $T \in \mathcal{N}(j)$ **do**
    **for all** $i \in T \setminus \{j\}$ **do**
      $\hat{V}^* \leftarrow$ **minloc**$(i, T, X_\mathrm{d})$
      **if** $\hat{V}^* < \hat{K}_i$ **then**
        Update key of $i$ to $\hat{V}^*$ in $Q$

### B. Local minimization problem

To satisfy (4), **minloc** must return a solution to the local minimization problem

$$\hat{V}^* = \inf_{\alpha_j} \left\{ \sum_{j \in T_i} \alpha_j \hat{V}_j + \left\| x_i - \sum_{j \in T_i} \alpha_j x_j \right\| \right\} \tag{5}$$

for any given $i$ and $T$, subject to linear constraints $\alpha_j \geq 0$ for all $j \in T_i$ and $\sum_{j \in T_i} \alpha_j = 1$. Note that the local minimization problem is equivalent to the shortest path problem between vertex $x_i$ and the proper face of simplex $T$ opposite $x_i$. The terminal cost on the face is given by linear interpolation of values $\hat{V}_j$ at vertices previously computed. We propose a geometric algorithm to solve (5) exactly for simplices of any dimension; see Algorithm 2 for details.

**Algorithm 2** Function **minloc**

**Input:** Vertex $x_i$, simplex $T$, vertex coordinates $X_\mathrm{d}$
**Output:** Solution to minimization problem (5)

Restrict $T$ to face $J \subset T$ such that all $\hat{V}_j$ for $j \in J$ are known
Let $\hat{V}' = \max_{j \in J} \hat{V}_j$ and $j' = \arg\max_{j \in J} \hat{V}_j$
Calculate normal vector $\vec{n}$ to planar section of $\hat{V}'$ level set of cost-to-go function (Fig. 1).
Calculate distance vector from $x_i$ to plane orthogonal to $\vec{n}$ and passing through $x_{j'}$ (Fig. 2).
**if** no barycentric coordinate of distance vector within simplex is negative **then**
  **return** $|\langle x_i - x_{j'}, \vec{n} \rangle| + \hat{V}'$
**else**
  Restrict $J$ to subset of non-negative barycentric coordinates and repeat from step 2

To interpret Algorithm 2, consider a two-dimensional simplex (triangle). Assume $x_3$ is a vertex with unknown $\hat{V}_3$, and without loss of generality, consider $\hat{V}_1 \leq \hat{V}_2$ to be known at vertices $x_1$ and $x_2$, respectively. In this setting, the problem is to find the shortest path from $x_3$ to the line segment between points $x_1$ and $x_2$, given a linear terminal cost $\hat{V}(x)$ such that $\hat{V}(x_1) = \hat{V}_1$ and $\hat{V}(x_2) = \hat{V}_2$.

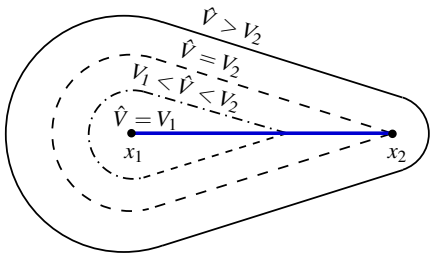


Fig. 1: Level sets of $\hat{V}(x)$ consist of two line segments bitangent to two circular arcs. One of the circular arcs is of zero radius if $\hat{V}(x) \leq \hat{V}_2$.

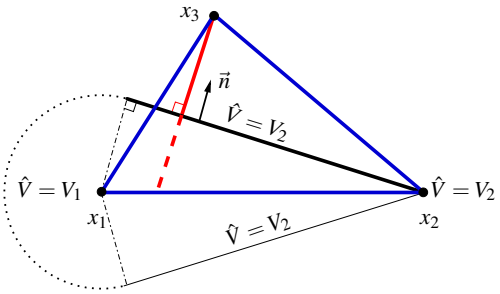


Fig. 2: The shortest path intersects the linear segment of the level set. In this case $\hat{V}_3 = \hat{V}_2 + \langle x_3 - x_2, \vec{n} \rangle$, in which $\langle \cdot, \cdot \rangle$ is a scalar product of two vectors.

For the considered shortest path problem, the level sets of the cost-to-go function $\hat{V}$ are illustrated in Fig. 1. Each level set consists of two line segments bitangent to two circular arcs, one of which may be of zero radius. Two cases are considered: $x_3$ belongs to a line segment or $x_3$ belongs to a circular arc. In the first case, the shortest path is orthogonal to the line segment of the level set $\{x \mid \hat{V}(x) = \hat{V}_2\}$; see Fig. 2. Hence, the solution is given by the distance to the line segment, $|\langle x_3 - x_2, \vec{n} \rangle|$, plus the cost-to-go function value on the segment, $\hat{V}_2$. In the second case, the shortest path terminates either at $x_1$ or at $x_2$; see Fig. 3. Thus, the solution to the local minimization problem is the lower of $\hat{V}_1 + \|x_3 - x_1\|$ and $\hat{V}_2 + \|x_3 - x_2\|$. Finally, consider the distance vector from vertex $x_3$ to the line embedding the linear segment of $\{x \mid \hat{V}(x) = \hat{V}_2\}$. This vector is within the triangle in the first case, and outside otherwise. Using barycentric coordinates of the distance vector, we have thus found a criterion to distinguish between the two cases considered.

### C. Simplicial A* algorithm

The SDA outlined in Section IV-A computes the approximate cost-to-go function in the entire environment regardless of the robot's initial configuration. If $x_\mathrm{I}$ is known, however, then it is desirable to perform costly computations only in the vicinity of the optimal path. In the discrete case, the A* graph search algorithm accomplishes this by employing a heuristic at each iteration of Dijkstra's algorithm [10]. We propose a Simplicial A* algorithm (SAA) by invoking a similar heuristic at each iteration of the SDA that narrows the focus of computations to vertices along the shortest path;

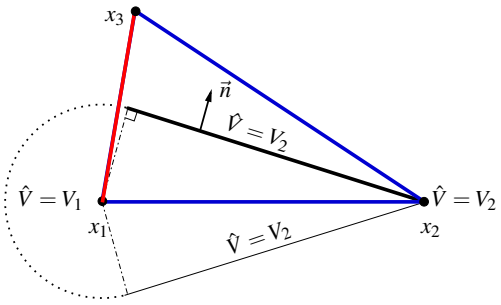


Fig. 3: The shortest path intersects the circular arc of the level set. In this case $\hat{V}_3 = \hat{V}_1 + \|x_1 - x_3\|$.

see Algorithm 3 for details.

**Algorithm 3** Simplicial A*

**Input:** Simplicial complex $(X_{\mathrm{d}}, \mathcal{T})$, goal set $X_{\mathrm{goal}}$, initial position of robot $x_{\mathrm{I}}$.

**Output:** Approximation of cost-to-go function, $\hat{V}_i$, in all vertices $x_i$ in neighborhood of optimal path.

$\hat{H}_i \leftarrow$ **heuristic**$(x_i, x_{\mathrm{I}})$ for all $x_i \in X_{\mathrm{goal}}$
Initialize priority queue $Q$ of all vertices. Set priority key $\hat{K}_i \leftarrow \hat{H}_i$, for all $x_i \in X_{\mathrm{goal}}$, and $\hat{K}_i \leftarrow \infty$, otherwise
**while** $Q$ is not empty **do**
  Pop $i$ with least key $\hat{K}_i$ from $Q$
  Set $\hat{V}_i \leftarrow \hat{K}_i - \hat{H}_i$
  **for all** $T \in \mathcal{N}(i)$ **do**
    **for all** $j \in T \setminus \{i\}$ **do**
      $\hat{V}^* \leftarrow$ **minloc**$(j, T, X_{\mathrm{d}})$
      $\hat{H}_j \leftarrow$ **heuristic**$(x_j, x_{\mathrm{I}})$
      **if** $\hat{V}^* + \hat{H}_j < \hat{K}_j$ **then**
        Update key of $j$ to $\hat{V}^* + \hat{H}_j$ in $Q$

Algorithm 3 is identical to Algorithm 1 in case of the trivial heuristic corresponding to no prior knowledge of the initial configuration. Although, if the heuristic approximates the *cost-to-come* function (i.e., the optimal cost of reaching the point from the initial state), then the SAA advances towards the initial configuration since $\hat{H}(x) + \hat{V}(x)$ is generally lower in this direction than in any other directions. Moreover, if the heuristic is admissible (defined in [10]) and consistent (covered in the next section), then the solution given by the SAA is identical to the solution given by the SDA at the evaluated vertices.

### D. Requirements on meshes

As with most mesh-based numerical methods, computational error and consistency depend crucially on mesh quality. Here, *computational error* is defined as the difference between the cost-to-go function and its discrete approximation. An algorithm is called *consistent* if it correctly solves the system of discrete dynamic programming equations. We show that simplex size affects computational error, and the regularity of simplex shape ensures consistency.

The computational error accumulates over the course of the SDA or the SAA. At each iteration of the algorithm linear interpolation error is introduced in the discrete dynamic programming principle. Consider (5) in two dimensions; the interpolation error is given by

$$\epsilon = \max_{\alpha_1, \alpha_2} \left| \hat{V}(\alpha_1 x_1 + \alpha_2 x_2) - \big(\alpha_1 \hat{V}(x_1) + \alpha_2 \hat{V}(x_2)\big) \right| . \quad (6)$$

The Taylor series expansion for $\hat{V}(x)$ in (6) suggests the asymptotic bound $\epsilon \sim \|x_1 - x_2\|^2$. Introducing the mesh quality parameter

$$h = \max_{T \in \mathcal{T}} \max_{i,j \in T} \|x_i - x_j\| , \quad (7)$$

we conclude that $\epsilon \sim h^2$.

The computational error, on the other hand, is proportional to the number of iterations times the interpolation error at each iteration, due to the error accumulation. The number of iterations is given by the number of vertices $M$ along the longest of all optimal paths, so that $E \sim M\epsilon$. We estimate $M$ by dividing the length of the longest of all optimal paths, $L$, by the mesh size $h$, which gives $E \sim Lh$, i.e., the computational error is linearly proportional to the mesh quality parameter $h$.

Consistency of the SDA is guaranteed, provided the discrete Bellman's principle satisfies the *causality property*: for any $i$ and $j$ sharing a simplex, value $\hat{V}_i$ depends on value $\hat{V}_j$ if $\hat{V}_j \leq \hat{V}_i$. This property parallels the consistency condition for Dijkstra's algorithm, i.e., edge weights must be positive. For the SDA, the causality property is satisfied if the following holds for (5):

$$\hat{V}^* > V_j \text{ , for all } j \text{ such that } \alpha_j > 0 . \quad (8)$$

The acute simplicial discretization guaranties (8). We demonstrate this for the two-dimensional case in the setting of the geometric construction from Section IV-B. First, notice that $\hat{V}^* \geq \hat{V}_1$. Second, $\hat{V}^*$ depends on $\hat{V}_2$ only if vertex $x_3$ belongs to a linear segment of the corresponding level set. Hence, it follows from Fig. 2 that $\hat{V}^* \geq \hat{V}_2$ if the projection of $x_3 - x_2$ on $\vec{n}$ is positive, which holds if the angle between edges incident at vertex $x_3$ is acute. Thus, as in [1], in 2D the consistency conditions are guaranteed if the triangulation is acute. Moreover, this geometric argument extends to higher dimensions, in which case we say a discretization is acute if the angles between all pairs of incident edges are acute.

Consistency of the SAA is implied by the modified causality property: for any $i$ and $j$ sharing a simplex, the value $\hat{V}_i$ depends on the value $\hat{V}_j$ if

$$\hat{H}_i + \hat{V}_i \geq \hat{H}_j + \hat{V}_j , \text{ or } \hat{H}_j - \hat{H}_i \leq \hat{V}_i - \hat{V}_j . \quad (9)$$

In (9) we replace $\hat{V}_i - \hat{V}_j$ with its minimum provided $\hat{V}_i$ depends on $\hat{V}_j$. In the two-dimensional case, the minimum is achieved if $\vec{n}$ is parallel to the side opposite $x_j$; see Fig. 2. Hence,

$$\hat{V}_i - \hat{V}_j \leq \|x_i - x_j\| \cos(\alpha) , \quad (10)$$

in which $\alpha$ is the angle between edges incident at vertex $x_i$. It follows from (9) and (10) that

$$\hat{H}_j - \hat{H}_i \leq \|x_i - x_j\| \cos(\alpha) \quad (11)$$

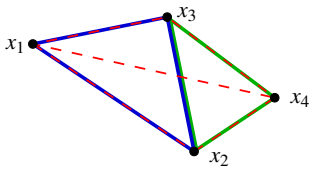


Fig. 4: Virtual edge flip trick in 2D.

must be satisfied for consistency. If $H$ satisfies (11), then it is called *consistent*. Higher-dimensional cases are analogous, except that $\alpha$ must be replaced with the maximum angle (minimum cosine) between any two edges incident at vertex $x_i$. In addition to the mesh requirements for the SDA, the heuristic must be consistent for consistency of the SAA.

The *airline distance* is a commonly used heuristic for the A* algorithm [10], but it fails to satisfy (11). Nevertheless, a rescaled airline distance provides a consistent heuristic. We introduce a mesh quality parameter

$$\gamma = \min_{T \in \mathcal{T}} \min_{i,j,k \in T} \cos(\angle(x_i, x_j, x_k)) \,, \tag{12}$$

in which $\angle(x_i, x_j, x_k)$ is the radian measure of the angle between vectors $x_i - x_j$ and $x_k - x_j$. The parameter $\gamma$ measures the regularity of the simplicial mesh, with $\gamma < 0$ indicating there is at least one non-acute simplex and $\gamma = 1/2$ for the "perfect" equilateral triangulation. It follows from the triangle inequality and (12) that the airline distance multiplied by $\gamma$ satisfies (11). Furthermore, as the mesh regularity improves, the parameter $\gamma$ increases, and the rescaled heuristic becomes increasingly usable.

If parameter $\gamma$ is small, then the rescaled airline distance is closer to the trivial heuristic, and the SAA has very little advantage over the SDA. Nevertheless, the rescaling coefficient and hence the quality of the heuristic can be improved significantly by implementing a virtual edge flip [1]. Figure 4 illustrates the idea of the virtual edge flip in 2D: the local minimization problem at vertex $x_1$ is solved for the red simplices instead of the blue simplex. Red simplices are built using vertex $x_4$, which is opposite face $(x_2, x_3)$ within the green simplex (blue and green simplices are required to share the face $(x_2, x_3)$). Using a virtual edge flip we can improve $\gamma$ for an equilateral triangulation up to $\sqrt{3}/2$. In higher dimensions the improvement is less pronounced, but three-dimensional experiments show that a virtual edge flip still provides reasonable $\gamma > 1/2$.

All the requirements on meshes can be summarized in the following two propositions.

*Proposition 1 (Accuracy):* For the proposed interpolation-based algorithms, the interpolation error is of the second order (i.e., $\epsilon \sim h^2$), and the computational error is of the first order (i.e., $E \sim h$), with respect to mesh quality parameter $h$ defined in (7).

*Proposition 2 (Consistency):* If the simplicial decomposition is acute, then the SDA is consistent. If additionally $\hat{H}$ satisfies $|\hat{H}_i - \hat{H}_j| \leq \gamma \|x_i - x_j\|$, for $\gamma$ as in (12), or, more generally, $\hat{H}$ satisfies (11), then the SAA is consistent.

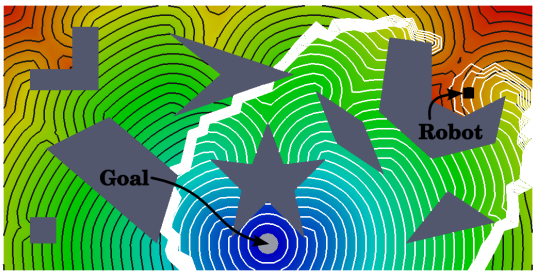


Fig. 5: Level sets of $\hat{V}$ in the 2D environment with obstacles (gray) computed using the SDA (black) and the SAA (white).

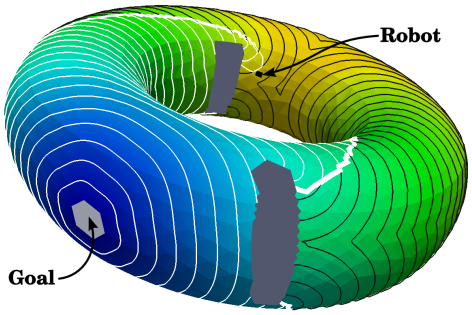


Fig. 6: Level sets of $\hat{V}$ on a torus with obstacles computed using the SDA (black) and the SAA (white).

## V. RESULTS AND DISCUSSION

The proposed algorithms were tested in three different scenarios: 1) a two-dimensional environment, 2) a two-dimensional manifold, and 3) a three-dimensional environment. In all test cases, polygonal obstacles were introduced. Simplicial meshes were generated using Gmsh software [9]. The same algorithms were applied, regardless of a problem's dimensionality or topology.

Figure 5 shows level sets of the approximate cost-to-go function in the 2D environment with obstacles. The black level sets are computed using the SDA, and the white level sets are computed using the SAA. The thick white line surrounds vertices computed by the SAA. As we can see, implementing a heuristic focuses the SAA on vertices primarily in the direction of the robot's location. Moreover, values of $\hat{V}$ at vertices computed by the SAA are identical to those computed by the SDA, and level sets coincide. Hence, the resulting optimal paths are identical.

Level sets of $\hat{V}$ computed on a 2D torus with obstacles are shown on Fig. 6. From this experiment, it is evident that the proposed interpolation-based approach generalizes to finding shortest paths (geodesics) on manifolds. As expected, the SAA outperforms the SDA by exploring fewer vertices.

In Fig. 7 two slices of level sets of $\hat{V}$ for a 3D environment with obstacles are illustrated. The black level sets correspond to SDA computations and white level sets correspond to SAA computations. As we can see, both algorithms extend to 3D cases. Moreover, the SDA explores more vertices than the SAA, and the trend remains in higher dimensions.

To compare the performance of the SDA vs. the SAA,

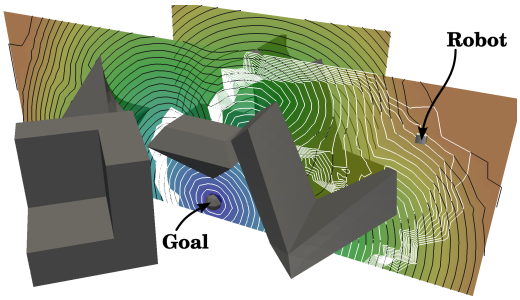


Fig. 7: Level sets of $\hat{V}$ in a the 3D environment with obstacles computed using the SDA (black) and the SAA (white).

TABLE I: Performance of SDA and SAA

| Experiment | Measure | SAA | SDA |
|---|---|---|---|
| 2D obs. | **minloc** call # | 15202 | 31314 |
| | vertex # | 727 | 1539 |
| | running time (sec) | 2.16 | 4.16 |
| 2D torus | **minloc** call # | 274240 | 557112 |
| | vertex # | 11550 | 23395 |
| | running time (sec) | 72.91 | 138.35 |
| 3D obs. | **minloc** call # | 837297 | 1675890 |
| | vertex # | 4515 | 9693 |
| | running time (sec) | 131.82 | 426.03 |

we introduce two performance measures: the number of **minloc** function calls and the number of computed vertices. The former is a better metric since the local minimization problem is computationally expensive. The latter, however, is adequate in case memory is limited. As we can see from Table I, the SAA consistently outperforms the SDA in both categories introduced above for all experiments considered. The running time of a Python implementation on Intel Core i7 3GHz is also illustrated in Table I.

## VI. CONCLUSIONS

In summary, we have developed an interpolation-based method for approximating the cost-to-go function associated with the Euclidean shortest path problem over a simplicial complex. We introduced simplicial versions of Dijkstra's algorithm and the A* algorithm to compute the approximate cost-to-go function from the system of the discrete dynamic programming equations efficiently. We have shown that both algorithms find a first-order accurate solution when provided with an acute simplicial complex, and a consistent and admissible heuristic in case of the SAA.The key features of the proposed framework are:

- The implementation is independent of the dimension or topology of the environment.
- For a simplicial complex with $N$ vertices, both algorithms have asymptotic running time $\mathcal{O}(N \log N)$.
- The SAA consistently explores fewer vertices and requires fewer **minloc** function calls then the SDA for the same simplicial complex.

*Acknowledgments*

This work is supported in part by and NSF grant 0904501 (IIS Robotics), NSF grant 1035345 (CNS Cyberphysical Systems), DARPA SToMP grant HR0011-05-1-0008, and MURI/ONR grant N00014-09-1-1052.

## REFERENCES

[1] T. Barth and J. A. Sethian, "Numerical Schemes for the Hamilton-Jacobi and Level Set Equations on Triangulated Domains," *Journal of Computational Physics*, vol. 145, no. 1, pp. 1–40, Sept. 1998.

[2] D. P. Bertsekas, *Dynamic Programming & Optimal Control, Vol. I*, 3rd ed. Athena Scientific, May 2005.

[3] J. Canny and J. Reif, "New lower bound techniques for robot motion planning problems," in *Proceedings of 28th Annual Symposium on Foundations of Computer Science*, Oct. 1987, pp. 49–60.

[4] J. Choi, J. Sellen, and C. K. Yap, "Approximate euclidean shortest path in 3-space," in *Proceedings of the tenth annual symposium on Computational geometry*, ser. SCG '94. New York, NY, USA: ACM, 1994, pp. 41–48.

[5] K. Daniel, A. Nash, S. Koenig, and A. Felner, "Theta*: Any-Angle Path Planning on Grids," *Journal of Artificial Intelligence Research*, vol. 39, pp. 533–579, 2010.

[6] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[7] D. Ferguson and A. Stentz, "Field D*: An Interpolation-Based Path Planner and Replanner," in *Robotics Research*, ser. Springer Tracts in Advanced Robotics, S. Thrun, R. Brooks, and H. Durrant-Whyte, Eds. Springer Berlin Heidelberg, 2007, vol. 28, ch. 22, pp. 239–253.

[8] A. F. Filippov, *Differential Equations with Discontinuous Righthand Sides: Control Systems (Mathematics and its Applications)*, 1st ed. Springer, Sept. 1988.

[9] C. Geuzaine and J.-F. Remacle, "Gmsh: A 3-d finite element mesh generator with built-in pre- and post-processing facilities," *International Journal for Numerical Methods in Engineering*, vol. 79, no. 11, pp. 1309–1331, 2009.

[10] P. Hart, N. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, Feb. 1968.

[11] J. Hershberger and S. Suri, "Efficient computation of Euclidean shortest paths in the plane," in *Proceedings of 34th Annual Symposium on Foundations of Computer Science*, 1993, pp. 508–517.

[12] S. Kapoor, S. N. Maheshwari, and J. S. B. Mitchell, "An Efficient Algorithm for Euclidean Shortest Paths Among Polygonal Obstacles in the Plane," *Discrete & Computational Geometry*, vol. 18, no. 4, pp. 377–383, Dec. 1997.

[13] R. Kimmel and J. A. Sethian, "Computing geodesic paths on manifolds," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 95, no. 15, pp. 8431–8435, July 1998.

[14] S. Koenig and M. Likhachev, "Improved fast replanning for robot navigation in unknown terrain," in *Proceedings of IEEE International Conference on Robotics and Automation*, 2002, pp. 968–975.

[15] J. S. B. Mitchell, "Shortest paths among obstacles in the plane," in *Proceedings of the ninth annual symposium on Computational geometry*, ser. SCG '93. ACM, 1993, pp. 308–317.

[16] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou, "The Discrete Geodesic Problem," *SIAM Journal on Computing*, vol. 16, no. 4, pp. 647–668, 1987.

[17] C. Papadimitriou, "An algorithm for shortest-path motion in three dimensions," *Information Processing Letters*, vol. 20, no. 5, pp. 259–263, June 1985.

[18] M. Pivtoraiko and A. Kelly, "Fast and Feasible Deliberative Motion Planner for Dynamic Environments," in *Proceedings of International Conference on Robotics and Automation (ICRA)*, May 2009.

[19] E. Rimon and D. E. Koditschek, "Exact robot navigation using artificial potential functions," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 5, pp. 501–518, Oct. 1992.

[20] J. A. Sethian and A. Vladimirsky, "Ordered Upwind Methods for Static Hamilton–Jacobi Equations: Theory and Algorithms," *SIAM Journal on Numerical Analysis*, vol. 41, no. 1, pp. 325–363, 2003.

[21] J. N. Tsitsiklis, "Efficient algorithms for globally optimal trajectories," *Automatic Control, IEEE Transactions on*, vol. 40, no. 9, pp. 1528–1538, Aug. 1995.