Article

The International Journal of Robotics Research
1–29

DOI: 10.1177/0278364919855422
journals.sagepub.com/home/ijr

Towards optimal path computation in a simplicial complex

Subhrajit Bhattacharya

Abstract

Computing optimal path in a configuration space is fundamental to solving motion planning problems in robotics and autonomy. Graph-based search algorithms have been widely used to that end, but they suffer from drawbacks. We present an algorithm for computing the shortest path through a metric simplicial complex that can be used to construct a piece-wise linear discrete model of the configuration manifold. In particular, given an undirected metric graph, G, which is constructed as a discrete representation of an underlying configuration manifold (a larger "continuous" space typically of dimension greater than one), we consider the Rips complex, $\mathcal{R}(G)$, associated with it. Such a complex, and hence shortest paths in it, represent the underlying metric space more closely than what the graph does. Our algorithm requires only a local connectivity-based description of an abstract graph, $G = (V, E)$, and a cost/length function, $d : E \to \mathbb{R}_+$, as inputs. No global information such as an embedding or a global coordinate chart is required. The local nature of the proposed algorithm makes it suitable for configuration spaces of arbitrary topology, geometry, and dimension. We not only develop the search algorithm for computing shortest distances, but we also present a path reconstruction algorithm for explicitly computing the shortest paths through the simplicial complex. The complexity of the presented algorithm is comparable with that of Dijkstra's search, but, as the results presented in this paper demonstrate, the shortest paths obtained using the proposed algorithm represent the geodesic paths in the original metric space significantly more closely.

Keywords

optimal path planning, simplicial complex, search algorithm

1. Introduction

In this paper we propose a novel search algorithm, Basic S*, which uses simplicial complex as a discrete representation of a configuration space to compute shortest paths in it. Basic S*, like graph search algorithms such as Dijkstra's and A*, requires only local connectivity information. However, Basic S* computes a piece-wise linear approximation of the underlying configuration metric manifold by representing the space as a simplicial complex instead of a graph. Performing search in the simplicial complex allows Basic S* to compute paths that better approximate the exact optimal solution in the underlying metric manifold than paths that are restricted to a graph alone.

1.1. Related work

Computing shortest path in a configuration space is fundamental to solving motion planning problems in robotics and autonomous systems. Whereas continuous methods (methods that do not explicitly rely a discrete representation of the underlying configuration manifold) for path planning do exist (Conner et al., 2003; Hsieh et al., 2007; Rimon and Koditschek, 1991; Zefran, 1996), they suffer from drawbacks (especially in the presence of obstacles/holes in the configuration space) such as difficulty in imposing arbitrary optimality criteria (potential/vector field methods (Hsieh et al., 2007; Rimon and Koditschek, 1991)), large search space (variational methods (Desai and Kumar, 1999; L'Affitto and Sultan, 2010; Mellinger et al., 2012)), termination at a local optimum (Conner et al., 2003; Khosla and Volpe, 1988; Kim and Khosla, 1992; Rimon and Koditschek, 1991) owing to non-convex search spaces, and in general a lack of rigorous guarantees when the configuration space has an arbitrary topology (non-contractible spaces) and non-trivial geometry (non-convex, general metric spaces).

Department of Mechanical Engineering and Mechanics, Lehigh University, Bethlehem, PA, USA

Corresponding author:
Subhrajit Bhattacharya, Department of Mechanical Engineering and Mechanics, Lehigh University, 19 Memorial Drive West, Bethlehem, PA 18015, USA.
Email: sub216@lehigh.edu

1.1.1. Traditional graph search-based methods. A robust and popular alternative to the continuous approaches is the discrete approach of graph search-based planning. The basic idea behind the approach is to sample points from the configuration space, and construct a graph by connecting "*neighboring*" vertices with edges (representing actions taking the system from one sampled configuration to another). Any trajectory in the original configuration space is approximated by a path in the graph (Bondy and Murty, 2007). One can thus employ a variety of search algorithms that can find optimal paths inside a graph, such as Dijkstra's (Dijkstra, 1959), A* (Hart et al., 1968), D* (Stentz, 1995), ARA* (Hansen and Zhou, 2007), or R* (Likhachev and Stentz, 2008), to compute the optimal path in the graph from a start vertex to a goal vertex. Sampling-based algorithms such as probabilistic roadmaps (PRM) (Kavraki et al., 1996) and rapidly-exploring random trees (RRT) (LaValle and Kuffner, 2001) samples vertices in a large or high-dimensional configuration space to construct a graph representation of the configuration space to find paths in it. Such discrete approaches for motion planning are suitable for use in configuration spaces of arbitrary topology and geometry, comes with guarantees on algorithmic completeness, termination, and optimality in the graph (or bounds on suboptimality), and are generally fast. Such conveniences are precisely the reason that graph search-based approaches have been extremely popular in solving motion planning problems on real robotic systems such as motion planning for autonomous vehicles (Ferguson et al., 2008; Hong et al., 2009; Montemerlo et al., 2008; Urmson et al., 2008), planning for robotic arms such as PR2 in cluttered environments (Cohen et al., 2010), multi-robotic systems (Bhattacharya et al., 2010; Swaminathan et al., 2015), and motion planning for systems involving cables (Bhattacharya et al., 2015b; Kim et al., 2014).

A drawback of using discrete graph-based approaches that find paths constrained to the graph alone is that a graph, constituting a one-dimensional subset of the original configuration space,[1] has a metric induced by the ambient configuration space that is significantly different from the metric of the configuration space itself. This means that paths that are optimal in the graph need not be optimal in the original configuration space. This issue is typically not remedied by reducing size of the discretization (see Figure 1). While adding more edges (actions) with far neighbors can help, it increase the computational complexity, and adding longer actions require us to have a priori knowledge of the nature of geodesics or line segments connecting configurations that are far from each other (i.e., breaks the local nature of the algorithm). In this paper we remedy this problem by using a simplicial complex instead of a graph. A simplicial complex contains higher than one-dimensional entities (called *simplices*), and hence is a richer discrete representation of a configuration space than a graph. It is thus a more suitable choice for solving optimal motion planning problems when we desire to have general solutions that are not necessarily restricted to a graph. Furthermore, the description of a simplicial complex as a discrete representation of a configuration space is highly local in nature, described in terms of local connectivity information only.

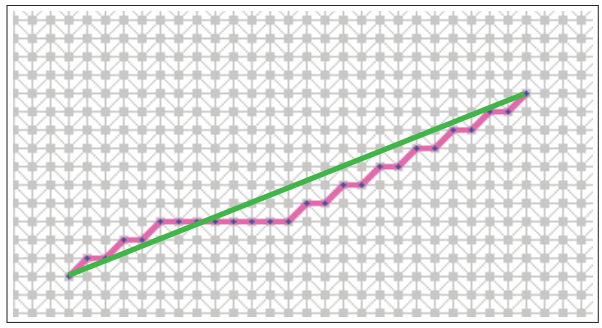


Fig. 1. The shortest path in the graph is longer than the shortest path in the original Euclidean plane.

1.1.2. Any-angle search algorithms. In recent years there have been significant effort in trying to remedy the problem of optimal paths in graphs being suboptimal in the original configuration space. Great strides have been made in multiple directions and for many specific classes of problems. All such approaches fall under the general category of what is known as "*any-angle path planning*" algorithms (Uras and Koenig, 2015). In the following sections we broadly classify the existing any-angle path planning algorithms in two groups and make comparison of those with Basic S* search algorithm.

1.1.3. Any-angle algorithms requiring a priori knowledge of line segments connecting distant points. One of the most successful any-angle path planning algorithms is Theta* (Nash et al., 2007, 2010). Given a discrete graph representation of a configuration space, Theta* is a fast, simple, yet powerful algorithm for finding optimal paths that can consist of line segments connecting vertices in the graph that are not necessarily neighbors (i.e., it constructs edges in the graph at an on-demand basis connecting vertices that are distant from one another). Theta* even works for environments with non-uniform cost functions (Daniel et al., 2010). However Theta* requires us to know what a "straight-line" segment connecting two distant vertices in the configuration space means, and that we have a way of quickly computing its length and checking its feasibility (i.e., requires "line of sight" information between a vertex and its far "grandparent"). While these are reasonable assumptions to make when a configuration space is presented as an embedding in an Euclidean space (i.e., a global coordinate chart is available and completely describes the configuration space), such assumptions may not be reasonable for more complex configuration spaces. The description of configuration spaces may be available only in terms of local connectivity (i.e., it may only be possible to identify valid configurations representing vertices that are close to each other

and, hence, establish edges between those, but nothing can be said a priori about the nature of the line segments or geodesic curves connecting configurations that are far from each other), and very often the configuration spaces with non-trivial topology may not even be embeddable in a subset of an Euclidean space of same dimension (for example an n-degree-of-freedom planar robot arm, the configuration space of which is an n-torus, cannot be embedded in an Euclidean space of the same dimension without creating *cuts* on it, in which case we end up throwing away paths that would have crossed the cuts or wrapped around the configuration space). Lazy Theta* (Nash et al., 2010), a variant of Theta*, also intermittently requires a priori information about long line segments/curves connecting distant points in the configuration space.

Likewise, in order to construct a visibility graph (Lozano-Pérez and Wesley, 1979; Jiang et al., 1993), it is necessary to have a global Euclidean embedding of the configuration space, as well as a priori understanding of shortest line segments that constitute common tangents between obstacles and tangents from start/goal points to the obstacles. In addition, at the construction stage of a visibility graph, those tangents do not take into account the underlying non-uniform traversal costs, and hence are not the optimal tangent curves. Within a specific homotopy class, Theta* will outperform the visibility graph approach, especially in the presence of non-uniform traversal costs. Similarly, Simple Sub-goal Graph (SSG) (Uras et al., 2013) is a graph representation of the free configuration space that relies on being able to construct straight line segments connecting distant points.

ANYA (Harabor and Grastien, 2013) is another any-angle path-finding algorithm that uses a square grid as a discrete representation of a planar domain and is guaranteed to return optimal paths on planar domains with uniform traversal cost. Recently, a generalization of ANYA called "Polyanya" has been proposed (Cui et al., 2017), where a navigation mesh (polygonal decomposition) is used instead of a uniform square grid. Like ANYA, Polyanya is optimal on planar domains with uniform traversal cost. The main advantage Basic S* offers over ANYA and Polyanya in two-dimensional settings is that it can deal with regions of non-uniform cost.

While all these any-angle algorithms perform extremely well when the configuration space is presented as a subset of an Euclidean space, and while algorithms such as Theta* can even deal with non-uniform traversal costs, all these algorithms require an explicit global embedding of the configuration space in an Euclidean space to be provided (i.e., a global coordinate chart is required, and the configuration space does not have non-trivial topologies) so that they know a priori what is meant by *line segments* connecting distant points in the environment. The Basic S* algorithm proposed in this paper, on the other hand, requires that we only know how each vertex in a discrete representation of the configuration space is locally connected to its neighboring vertices. It does not require us to know a priori what a line segment or a geodesic curve connecting distant points in the configuration space is, does not require us to check the feasibility of such a curve, nor compute its cost.

1.1.4. Cost-interpolation-based any-angle algorithms. The class of any-angle algorithms that Basic S* is ore closely related to is that of cost interpolation-based algorithms. Field D* (Ferguson and Stentz, 2007) is an algorithm that uses a uniform square grid discretization of subset of a plane, and computes the g-score of vertices in its *open set* by interpolating between the g-scores of its potential parents. Field D*, in its current form, is highly restrictive in terms of the type of discretization it can use (only uniform square grids on subsets of an Euclidean plane), relies on a global coordinate chat/embedding for computing the interpolations, and do not explicitly consider non-uniform traversal costs or anisotropic metric.

Simplicial Dijkstra's (Yershov and LaValle, 2011) and wavefront propagation (Xu et al., 2015) are algorithms based on simple linear interpolation in local g-score update. While much of these algorithms do not rely on the availability of a global Euclidean embedding or coordinate charts, and can work with more general discretization, currently these algorithms work only for two-dimensional manifolds only, with specific requirements about the mesh construction. However, what makes the algorithms proposed by Yershov and LaValle (2011) and Xu et al. (2015) less suitable for solving robot path planning problems is the fact that neither of these works explicitly computes the shortest paths connecting two points in the configuration space. Instead they compute the distance field to every point in the configuration space and stops short of computation of the paths. Furthermore, neither of these algorithms are accompanied by suboptimality estimates (compared with length of exact/true geodesic curves in the configuration manifolds) as we do for the Basis S* algorithm.

Closely related to cost-interpolation-based algorithms is Mitchell et al. (1987), which considers geodesic computation on polyhedral surfaces with Euclidean metric. In contrast, the Basic S* algorithm proposed in this paper is applicable to general simplicial complexes that are not necessarily two-dimensional. Furthermore, Basic S* does not require on-the-fly subdivision of the simplices. We also make a clear connection between the geodesic in the underlying configuration manifold and the shortest path computed by Basic S* in the discrete simplicial complex representation by providing a measure of suboptimality in the latter when compared against the former. We provide thorough evaluation of Basic S* in multiple different types of environments (including three-dimensional configuration spaces and spaces with non-Euclidean metric).

1.2. Features and assumptions of Basic S*

Broadly speaking, the Basic S* algorithm proposed in this paper falls under the category of any-angle path planning algorithms. However, our method is purely local (requiring only local connectivity information as in a discrete graph representation and does not require any embedding or coordinate chart), is suitable for estimating optimal solutions in the configuration spaces of arbitrary topology, geometry, and dimension, and allows explicit computation of the path along with theoretical estimate of suboptimality (compared with the length of exact/true geodesic curves in the underlying configuration manifolds). Instead of planning paths in a graph, we propose the *Basic S** algorithm for finding shortest paths through simplicial complexes. In particular, given a graph, we consider the *Rips complex* of the graph, and compute shortest path in that complex (Figure 2). Some features of our proposed method and comparison with other state-of-the-art are listed in the following.

- Our method **does not require the graph to be embedded in an Euclidean space, nor does it require a global coordinate chart**. The input required to our algorithm is the abstract graph, $G = (V, E)$ (where the vertices are abstract points or any type), and a cost/length function, $d : E \to \mathbb{R}_+$. In fact, the graph, G, need not be available in its entirety to begin with. All that the Basic S* algorithm requires is a *neighbor function*, $\mathcal{N}_G$, which computes the "neighbors" of a given vertex in G, and the cost of the edges connecting to them. Such a description is purely local in nature since local connectivity information (i.e., how a vertex is connected to its neighbors) is sufficient to describe the complete metric graph. Embeddings are constructed locally for simplices as required. This is in contrast to algorithms such as Theta* (Daniel et al., 2010; Nash et al., 2010), SSG (Uras et al., 2013), ANYA (Harabor and Grastien, 2013), and Field D* (Ferguson and Stentz, 2007).
- The only required input to our algorithm is a metric graph (i.e., a graph with specified edge costs/lengths). The input graph can be an arbitrary, abstract metric graph. In particular, we do not require the underlying metric space (whose discrete representation is the graph) to be a subset of flat/Euclidean space with the Euclidean metric (unlike what is required by Visibility graph (Lozano-Pérez and Wesley, 1979; Jiang et al., 1993) and ANYA (Harabor and Grastien, 2013)). Informally speaking, **our method can deal with graphs with "*non-uniform traversal costs*," both non-homogeneous and anisotropic**.
- Our algorithm is local, requiring the abstract graph, $G = (V, E)$, and the cost/length function, $d : E \to \mathbb{R}_+$, presented in terms of local connectivity information only. In particular, **for each vertex in the graph, Basic S* just needs to know what other valid vertices lie in its local neighborhood (the immediate neighbors in G, with which it establishes edges), and the costs/lengths of edges connecting to those**. This is described in terms of a *neighbor function*, $\mathcal{N}_G$, such that $\mathcal{N}_G(v),\ v \in V$, gives the set of vertices that are connected to v via an edge in E. Since this neighborhood is small for every vertex, if the edges connecting a vertex to its neighbors are valid (i.e., do not intersect obstacle), then the points inside the simplex in their interior are assumed to be valid as well. Basic S* does not require global information such as line of sight to distant vertices or a global embedding. In particular, this is unlike Theta* (Daniel et al., 2010; Nash et al., 2010), SSG (Uras et al., 2013), or Visibility graphs (Lozano-Pérez and Wesley, 1979; Jiang et al., 1993) where it is necessary to have a priori understanding of line segments or curves joining distant vertices.
- Our algorithm is designed for simplicial complexes of arbitrary dimensions and **does not require any specific kind of discretization, as long as the simplicial complex covers the entire original configuration space** (i.e., there is a bijection between the configuration space and the simplicial complex representation of it). In particular, for any two-dimensional configuration space, any arbitrary triangulation is sufficient. This is in contrast to, for example, Field D* (Ferguson and Stentz, 2007) which relies on a square grid discretization of a plane. Simplicial Dijkstra's (Yershov and LaValle, 2011) also has strict requirements on the types of discretization that it can deal with.
- We consider an **accurate geometric model in computing/extrapolating the distances based on local embedding** of simplices in a model Euclidean space. This is in contrast to Simplicial Dijkstra's (Yershov and LaValle, 2011) or wavefront methods (Xu et al., 2015), where an artificial local minimization problem is solved. The accurate model allows us to guarantee that the cost/length of shortest paths computed using the proposed algorithm approaches the true geodesic distance on Riemannian manifolds as the discretization size is made finer.
- We also propose a ***path reconstruction*** **algorithm** to construct the shortest paths in the configuration space. This, once again, is in contrast to Simplicial Dijkstra's (Yershov and LaValle, 2011) or wavefront methods (Xu et al., 2015), where the authors simply compute the value of the distances to points in the environment, stopping short of explicit computation of the shortest paths.
- While the input to our algorithm is a metric graph, the underlying structure on which we attempt to compute an optimal path is a metric simplicial complex (for a given graph we consider its Rips complex). More generally, **our algorithm can be used to compute shortest paths in metric simplicial complexes (not necessarily a Rips complex of a metric graph)**.

- We have also provided **theoretical results on the suboptimality/error in the cost of the path computed** by Basic S* (compared with the true/exact geodesic path in the underlying manifolds) and have shown that this suboptimality goes down quadratically fast as the size of the simplices in the discrete representation are decreased.

Our focus is the development of an algorithm that can compute paths that are close to the optimal geodesics in metric configuration manifolds represented by a metric graph constructed by sampling points from the manifold. However, we want the computed path to be not restricted to the graph alone so that it closely represents the true geodesic path in the underlying metric space. Basic S* uses a piece-wise linear approximation of the underlying metric manifold by approximating the space as a metric simplicial complex (constructed as the Rips complex of the given metric graph representation). While optimality of the path computed using Basic S* is limited by the size/fineness of this discrete representation of the manifold, we demonstrate that it is possible to approximate the metric of a Riemannian manifold (possibly with boundaries) with arbitrary precision by increasing the *fineness* of the discrete representation. As a consequence, we call the paths computed by Basic S* on a simplicial complex representation of a Riemannian manifold to be "(arbitrarily)-close-to-optimal," paths that can be made to match the optimal path on the manifold arbitrarily closely by reduction of the discretization size.

Basic S* is local in nature: for each vertex in the graph, Basic S* just needs to know what other valid vertices lie in its local neighborhood, i.e., the immediate neighbors of that vertex in the graph, G. Basic S* encompasses/combines features (e.g., being local nature and not requiring a priori knowledge of line segments connecting distant points, no requirement of global embedding/coordinate chart, being suitable for arbitrary dimension and topology, and being able to reconstruct the close-to-optimal paths) that are not present all at once in any of the other algorithms. We use the Dijkstra's search as the backbone for our algorithm, and develop techniques to incorporate simplicial data into it. More efficient versions of the algorithm (incorporating features of heuristic, randomized, incremental, and any-time search algorithms) are within the scope of future work.

As will be discussed in the complexity analysis (Section 4.3), in a simplicial complex with acute simplices in which node reopening is guaranteed not to occur, the time complexity of Basic S* is $O(|V|\log|V|)$ for a graph with constant average degree. This is the same time complexity as Dijkstra's search on a graph with constant average degree (the time complexity for Dijkstra's is $O((|E|+|V|)\log(|V|))$, but with constant average degree for every vertex, $|E| = k|V|$ (Cormen et al., 2009)). However, even assuming that the dimension of configuration manifold is kept fixed, Basic S* needs to perform more computation per vertex (construction of the simplices, computation of Euclidean embedding, $\overline{d}$-value extrapolation) than an algorithm such as Dijkstra's. So usually Basic S* will have slower runtime than algorithms such as Theta* and Dijkstra's. To that end we highlight that the strengths of Basic S* instead lie in its versatility rather than in its runtime: it is an any-angle type algorithm that is applicable to configuration manifolds of *arbitrary dimensions* (not just two-dimensional domains), configuration manifolds of *arbitrary topology* (not just subsets of planar domains, such as in case of a robot arm and topological path planning), *non-uniform traversal costs*, and *does not require line-of-sight information* (i.e., a priori knowledge of "line segments" connecting distant points, as required by algorithms such as Theta*). All these features that are not simultaneously satisfied by any existing algorithm.

1.3. Outline of paper

In the next subsection we introduce some preliminary notation and definitions. Following that, we introduce the main Basic S* algorithm, the subprocedures involved in it, and a path reconstruction algorithm. Some theoretical analyses follow. Finally we present simulation results. For better readability, many of the detailed proofs and derivations have been moved to the appendix.

1.4. Preliminaries

In this section we present the combinatorial definitions of a simplex and a simplicial complex. A simplicial complex can be thought of to be a higher-dimensional generalization of a graph: instead of having only zero- and one-dimensional entities (vertices and edges, respectively), a simplicial complex can contain general n-dimensional entities (called an *n-simplex*) for $n = 0, 1, 2, \ldots$. Such a complex can be used as a discrete representation of a configuration space. For example, in a two-dimensional configuration space, a simplicial complex can be interpreted as a triangulation of the space, with the triangular elements being the 2-simplices. In a three-dimensional configuration space it is a mesh with tetrahedral elements (where the tetrahedra are the 3-simplices). The formal definitions are as follows.

Definition 1 (Simplicial complex (Hatcher, 2001): combinatorial definition)**.** A simplicial complex, $\mathcal{C}$, constructed over a set V (the *vertex set*) is a collection of sets C_n, $n = 0, 1, 2, \ldots$, such that the following hold.

(i) An element in C_n, $n \geq 0$, is a subset of V and has cardinality $n+1$ (i.e., for all $\sigma \in C_n$, $\sigma \subseteq V$, $|\sigma| = n+1$). Here σ is called a "*n-simplex*."
(ii) If $\sigma \in C_n$, $n \geq 1$, then $\sigma - v \in C_{n-1}$, $\forall v \in \sigma$. Such a $(n-1)$-simplex, $\sigma - v$, is called a "*face*" of the simplex σ.

The simplicial complex is the collection $\mathcal{C} = \{C_0, C_1, C_2, \ldots\}$. We also define $C_* = C_0 \cup C_1 \cup C_2 \cup \cdots \subseteq \mathcal{P}(V)$.

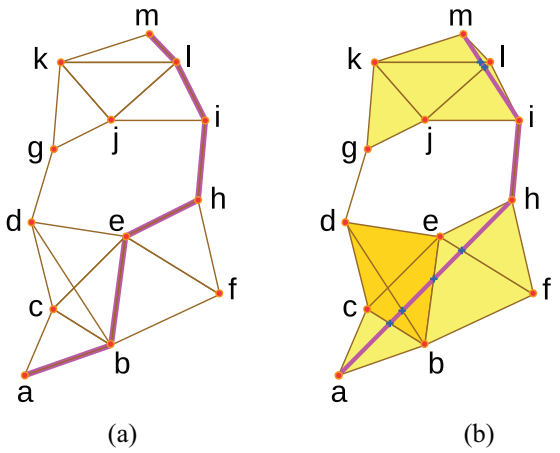


Fig. 2. (a) Graph $G = (V, C_1 = E)$ (with vertex set V and edge set E), and the shortest path in G connecting $\mathsf{a}, \mathsf{m} \in V$. (b) The Rips complex $\mathcal{R}(G) = \{C_0, C_1, C_2, C_3, \ldots\}$, and the shortest path in (an Euclidean embedding of) $\mathcal{R}(G)$ connecting $\mathsf{a}, \mathsf{m} \in V$. In this example, $V = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}, \mathsf{f}, \mathsf{g}, \mathsf{h}, \mathsf{i}, \mathsf{j}, \mathsf{k}, \mathsf{l}, \mathsf{m}\}$ is the vertex set, $C_0 = \{\{\mathsf{a}\}, \{\mathsf{b}\}, \{\mathsf{c}\}, \{\mathsf{d}\}, \{\mathsf{e}\}, \{\mathsf{f}\}, \{\mathsf{g}\}, \{\mathsf{h}\}, \{\mathsf{i}\}, \{\mathsf{j}\}, \{\mathsf{k}\}, \{\mathsf{l}\}, \{\mathsf{m}\}\}$ is the set of 0-simplices (colored in red), $C_1 = \{\{\mathsf{a}, \mathsf{b}\}, \{\mathsf{b}, \mathsf{c}\}, \{\mathsf{c}, \mathsf{a}\}, \{\mathsf{c}, \mathsf{d}\}, \{\mathsf{c}, \mathsf{b}\}, \{\mathsf{b}, \mathsf{d}\}, \{\mathsf{d}, \mathsf{e}\}, \{\mathsf{e}, \mathsf{b}\}, \{\mathsf{e}, \mathsf{c}\}, \{\mathsf{b}, \mathsf{f}\}, \{\mathsf{e}, \mathsf{f}\}, \{\mathsf{f}, \mathsf{h}\}, \{\mathsf{e}, \mathsf{h}\}, \{\mathsf{h}, \mathsf{i}\}, \{\mathsf{d}, \mathsf{g}\}, \{\mathsf{g}, \mathsf{j}\}, \{\mathsf{j}, \mathsf{k}\}, \{\mathsf{j}, \mathsf{l}\}, \{\mathsf{k}, \mathsf{l}\}, \{\mathsf{j}, \mathsf{i}\}, \{\mathsf{i}, \mathsf{l}\}, \{\mathsf{l}, \mathsf{m}\}, \{\mathsf{k}, \mathsf{m}\}\}$ is the edge set (the set of 1-simplices; colored in brown), $C_2 = \big\{\{\mathsf{a}, \mathsf{b}, \mathsf{c}\}, \{\mathsf{b}, \mathsf{c}, \mathsf{d}\}, \{\mathsf{b}, \mathsf{c}, \mathsf{e}\}, \{\mathsf{e}, \mathsf{c}, \mathsf{d}\}, \{\mathsf{b}, \mathsf{e}, \mathsf{d}\}, \{\mathsf{e}, \mathsf{b}, \mathsf{f}\}, \{\mathsf{f}, \mathsf{h}, \mathsf{e}\}, \{\mathsf{i}, \mathsf{j}, \mathsf{l}\}, \{\mathsf{j}, \mathsf{l}, \mathsf{k}\}, \{\mathsf{k}, \mathsf{l}, \mathsf{m}\}, \{\mathsf{k}, \mathsf{g}, \mathsf{j}\}\big\}$ is the set of 2-simplices (colored in light yellow), and $C_3 = \{\{b, c, d, e\}\}$ is the set of 3-simplices (colored in darker yellow).

In general, a n-simplex is a set containing $n+1$ elements, $\sigma = \{v_0, v_1, v_2, \ldots, v_{n-1}, v_n\}$, where $v_i \in V$, $i = 0, 1, \ldots, n$.

In algebraic topology, one imparts group or vector space structures on these sets via operation completions, and defines linear maps between those (the *boundary maps*). However, for the purpose of this paper we do not require such algebraic constructions.

A Rips complex is special type of simplicial complex constructed from an input graph in which every *clique* is "filled-in" with a simplex. The formal description follows.

Definition 2 (Rips complex of a graph, $\mathcal{R}(G)$)**.** If G is an undirected graph with V its vertex set and E its edge set, we define the *Rips complex of the graph*, $\mathcal{R}(G)$, to be the simplicial complex with an n-simplex consisting of every $(n+1)$-tuple of vertices that are all connected to each other (a *clique*). We use the notation $\mathcal{R}(G) = \{C_0, C_1, C_2, C_3, \ldots\}$ is a simplicial complex such that $C_0 = \{\{a\} \mid a \in V\}$ and for $\sigma \in C_n$, $n > 1$, and $a, b \in \sigma$, we have $\{a, b\} \in E$. In addition, define $C_* = C_0 \cup C_1 \cup C_2 \cup \cdots$.

The set C_0 is the set of 0-simplices consisting of singleton sets, each containing a single vertex from V, while the set $C_1 = E$. From now on, whenever we refer to a simplicial complex, unless otherwise specified, we will refer to the Rips complex $\mathcal{R}(G) = \{C_0, C_1, C_2, \ldots\}$, for a given graph $G = (C_0, C_1)$. Figure 2 illustrates the Rips complex of a graph with an explicit example of the sets $C_0, C_1, C_2, \ldots$.

A function, $d : C_1 \to \mathbb{R}_+$, defining "lengths" of the 1-simplices provides an unique Euclidean embedding (up to rotations and translations) of every simplex in the complex, thus inducing a metric in the complex. This is what we refer to as a *metric simplicial complex*.

Definition 3 (Acute simplex)**.** Consider an n-dimensional metric simplex, $\sigma \in C_n$, with an embedding in an n-dimensional Euclidean space (an explicit construction of such an embedding will be discussed in Proposition 1). The simplex is called acute if all its *dihedral angles* (angles between its faces) are less than $\pi/2$ (Kopczynski et al., 2012).

A consequence of the above definition is that in an acute simplex the angle between any two adjacent edges (1-subsimplices of σ that share a vertex) is acute. As a result, if H is an $(n-1)$-dimensional hyperplane passing through any vertex $v \in \sigma$, and $H^\perp$ is the normal to the hyperplane at v, then the interior of the simplex cannot intersect both H and $H^\perp$.

1.5. An overview of the Basic S* algorithm

Basic S* search works with a discrete representation of a configuration space analogous to Dijkstra's or A*. However, instead of a graph, Basic S* uses a simplicial complex as the discrete representation (we will usually use the Rips complex of a graph embedded in the configuration space). Once the discrete representation is constructed, the algorithm is indifferent to the original configuration space, and simply ensures that the computed path lies inside the discrete representation (just as A* or Dijkstra's restricts paths to the graph, the Basic S* restricts paths to the Rips complex). Intuitively, the Rips complex of a graph is a construction that "fills in" the triangles, tetrahedra, and higher-dimensional simplices that are present in a graph. This creates a piece-wise linear approximation of the original (free) configuration space, and Basic S* computes the shortest path in this complex.

The first step in Basic S*, like any search algorithm, requires the computation of distance from the start to other vertices (0-simplices) in the complex (we call this distance function $\overline{d}$-distance). The principle behind this computation is to extrapolate the $\overline{d}$-distance to the apex, u, of a simplex, given the $\overline{d}$-distance to the vertices that constitute its "base" of the simplex and the edge lengths of the simplex. We explore two different extrapolation methods (spherical and linear). The second step in the algorithm is path reconstruction, which involves the computation of the gradient of $\overline{d}$ through the simplicial complex.

Algorithm 1: Basic S*

$(\bar{d}, cfp)$ = *Basic_S** (G, d, s)
Inputs: a. Graph G with vertex set V and (undirected) edge set $E = C_1$ presented in terms of the neighbor function, $\mathcal{N}_G$
b. A length/cost function on the edge set, $d : E \to \mathbb{R}_+$
c. Start vertex, $s \in V$.
Outputs: a. The distances from s to every vertex in the graph, $\bar{d} : V \to \mathbb{R}_+$
b. Came-from-point map, $cfp : V \to C_* \times \mathbb{R}^k$

1 Implicitly assume $\bar{d}(v) := \infty$ for any vertex v that has not been generated/visited.
2 Implicitly assume $cfp(v) := \emptyset$ for any vertex v that has not been generated/visited.
3 Set $\bar{d}(s) \leftarrow 0$ // start vertex.
4 Set $Q := \{s\}$ // Set of open vertices.
5 **while** ($Q \neq \emptyset$ AND stopping criterion not met)
6 Set $q := \arg\min_{q' \in Q} \bar{d}(q')$ // Maintained by a heap data structure.
7 Set $Q \leftarrow Q - \{q\}$ // Remove q from Q.
8 **for each** ($u \in \mathcal{N}_G(q)$) // For each neighbor of q (both expanded and open).
9 Set $S := \{y \mid y \in \mathcal{N}_G(q), y \in \mathcal{N}_G(u), y \notin Q\}$ // expanded common neighbors of q and u.
10 Set $\mathcal{MS} := MaximalSimplices_{\mathcal{R}(G)}(S; \{u, q\})$ // maximal simplices attached to $\{u, q\}$.
11 **for each** ($\sigma \in \mathcal{MS}$)
12 Set $\left(d', (\sigma', \overline{w}')\right) := DistanceThroughSimplex_{(d,\bar{d})}(\sigma, u)$ // distance to u through simplex σ.
13 **if** ($d' < \bar{d}(u)$) // Found better $\bar{d}$-score for this neighbor.
14 Set $\bar{d}(u) \leftarrow d'$
15 Set $cfp(u) \leftarrow (\sigma', \overline{w}')$
16 Set $Q \leftarrow Q \cup \{u\}$. // Will "reopen" u if it is already expanded.
17 **return** $\bar{d}$, cfp.

2. Basic S* algorithm

The idea behind a search algorithm in a simplicial complex is similar to the standard search algorithms such as Dijkstra's or A*, but instead of restricting paths to the graph G, it allows paths to pass through simplices of the complex. In the proposed algorithm, when expanding a vertex $q \in V$, and updating a particular neighbor $u \in V$, for computing the minimum distance of u from the start (we will refer to that distance as $\bar{d}$), instead of replacing the earlier values with the new lower value, we identify the *maximal simplices* (Definition 4) containing q, u, and other already expanded vertices. We then construct a local Euclidean embedding of each of the simplices, along with an embedding of the start vertex in the same Euclidean space (Figure 3d). This gives an accurate geometric model of the local simplex and lets us compute the $\bar{d}$-distance of u through the simplex (i.e., estimated length/cost of a path from the start vertex to u that can potentially intersect the face of the simplex opposite to u instead of having to pass through the vertex q). These features (identifying maximal simplices and constructing accurate local embedding) are distinct from prior simplicial search algorithms such as those in Xu et al. (2015) and Yershov and LaValle (2011). Furthermore, the local construction requirement means that $\mathcal{R}(G)$ is constructed on the fly simply by looking at the connectivity in the neighborhood of u, and the algorithm does not require the entire complex to be stored in memory to start with. The algorithm, which we call *Basic S**, is illustrated in the example of Figure 3. The complete pseudocode for the algorithm (with a single start vertex and no specified goal vertex) is given in Algorithm 1. Specific details on the subprocedures in the pseudocode appear in subsections under the present section.

The most important subroutines in this algorithm are the *MaximalSimplices* and the *DistanceThroughSimplex* procedures. Much of the remainder of this section will be devoted towards describing those. But before that we highlight a few other key features of the algorithm.

1. Similar to Dijkstra's or A* search, we maintain a list of open (*unexpanded*) vertices (the *open list*), Q, in a heap data structure with the $\bar{d}$-values (which, in the search algorithm literature, has be traditionally called the g-score) of the vertices being the heap keys. Q is initiated with the start vertex. At each of the while loop iterations (starting on line 5), the open vertex with lowest $\bar{d}$-value is popped ("expanded") and the $\bar{d}$-value of its neighbors are checked for improvement (loop starting on line 8).
2. Unlike Dijkstra's algorithm, however, we do not restrict the update only to open neighbors of q, and instead check for update for all neighbors, u, in $\mathcal{N}_G(q)$. This is because there may be obtuse simplices, that get generated at a later stage, through which the distance to an already expanded vertex may be shorter. We will refer

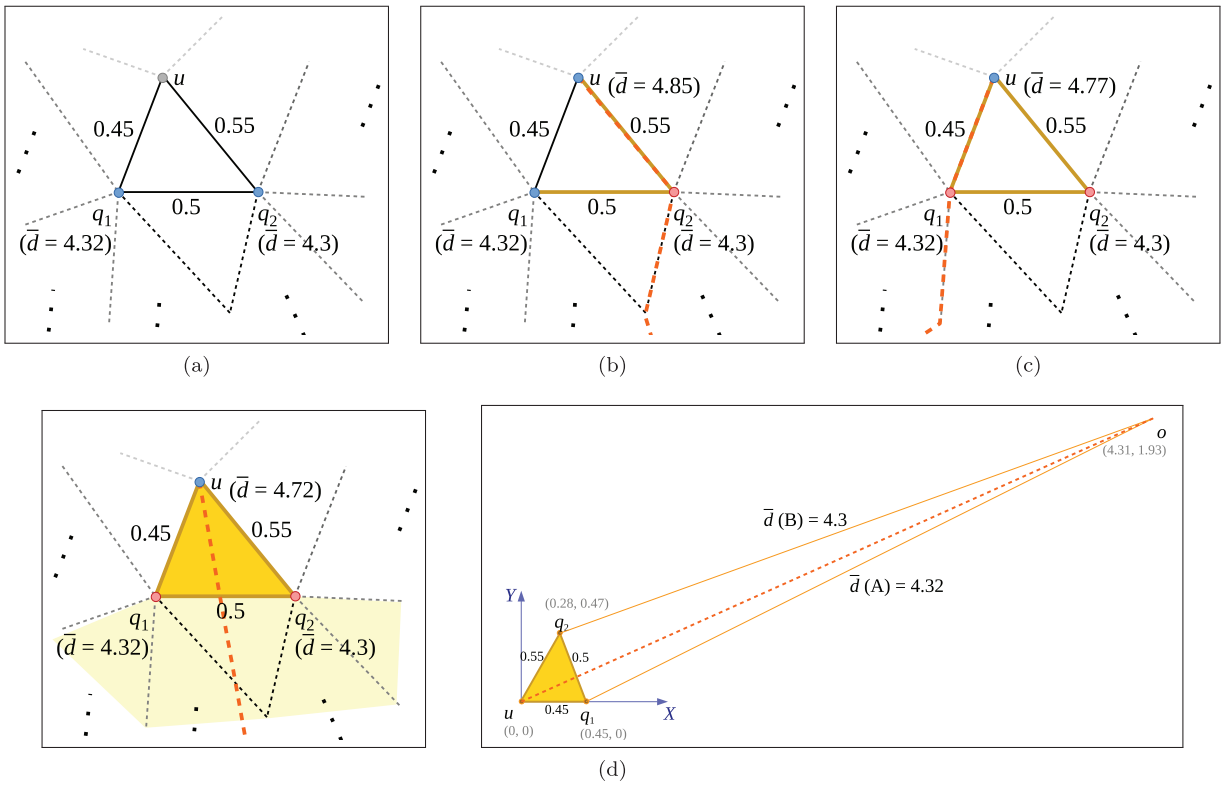


Fig. 3. Comparison between Dijkstra's search algorithm for graph, G, and the Basic S* search algorithm for $\mathcal{R}(G)$. (a) Vertices q_1 and q_2 are candidates for being *expanded*. (b) **Expansion of q_2 in Dijkstra's or Basic S*:** q_2 has a lower $g = \overline{d}$ value (distance from s) than q_1, hence is expanded first. The g-value of u is set to $\overline{d}(q_2) + d(u, q_2) = 4.3 + 0.55 = 4.85$: the distance to u through the edge $\{q_2, u\}$ (also a 1-simplex containing q_2 and u). The "came-from" vertex of u is set to q_2. This step, in this example, will be the same for both Dijkstra's algorithm and Basic S*. (c) **Expansion of q_1 in Dijkstra's:** Expansion of q_1 gives a lower g-value of u for a path through the edge $\{q_1, u\}$ (since $\overline{d}(q_1) + d(u, q_1) = 4.32 + 0.45 = 4.77$), hence its g-value is updated, and the "came-from" vertex of u is set to q_1. This is typical of a search algorithm such as Dijkstra's. (d) **Expansion of q_1 in Basic S*.** *Left:* In the Basic S* algorithm, when vertex q_1 is expanded, and the $\overline{d}$-value of u is to be checked for update, we construct all the *maximal simplices* attached to q_1, containing u, and containing already-expanded vertices. In this example $\{q_1, q_2, u\}$ is such a maximal 2-simplex (since q_2 has already been expanded). The distance to u through this 2-simplex is computed, and is the candidate for $\overline{d}$-value update. In this example, if the $\overline{d}$ values of q_1 and q_2 are once again $\overline{d}(q_1) = 4.32$ and $\overline{d}(q_2) = 4.3$, then the distance to u through $\{q_1, q_2, u\}$ is 4.72, and hence its $\overline{d}$-value is updated and the "came-from" simplex of u is set to $\{q_1, q_2, u\}$. *Right:* The way we compute the distance through the 2-simplex involves constructing an embedding of the abstract metric simplex, $\{q_1, q_2, u\}$, in a same-dimensional Euclidean plane and an embedding of the metric 2-simplex $\{q_1, q_2, o\}$ with the lengths of $q_1 o$ and $q_2 o$ being equal to the corresponding vertices' $\overline{d}$-value. Since $\overline{ou}$ intersects $\overline{q_1 q_2}$ in the embedding, the length of $\overline{ou}$ gives the desired value of distance to u through $\{q_1, q_2, u\}$. We refer to this method of computing the distance as *spherical extrapolation*. An alternative to this is *linear extrapolation* (refer to Figure 7). (a)–(c) shows steps in a typical graph search algorithm. In Basic S*, the steps (a) and (b) are the similar, but step (c) is replaced by the step (d).

to this process of updating an already-expanded vertex and putting it back to the *open list* as "reopening" to be consistent with the related search literature that describe a similar process (Felner et al., 2011). This is described in more detail in Figure 4. In Section 4.2 we however prove that Basic S* does not reopen vertices if all the simplices in the complex are acute, thus giving us a sufficient condition under which the algorithm is guaranteed to not reopen any expanded vertex.

3. The "while loop" terminates when either the open list is empty (all vertices in the connected component of G have been expanded), or when a stopping criterion is met. In simplicial complexes with all acute simplices the "stopping criterion" can simply be expansion of a goal vertex, since in such a complex vertices are guaranteed to be not reopened. However, in general, in complexes with obtuse simplices, this will not be sufficient since improvement of the goals $\overline{d}$-value is possible later through the process of vertex reopening. In that case, the Basic S* algorithm should expand beyond the goal vertex to check for reopening and re-expansion of the goal vertex. As discussed in Section 4.3, in general it may

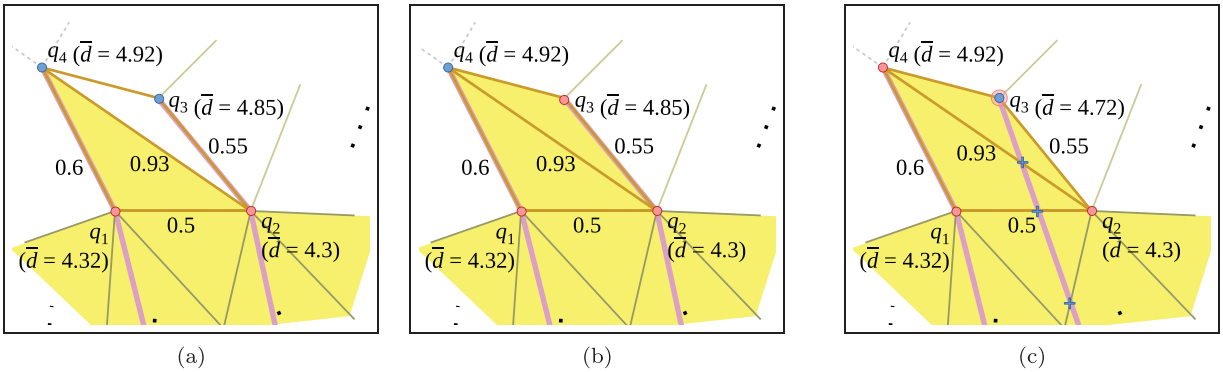


Fig. 4. Illustration of a scenario where improvement of the $\overline{d}$-value of a vertex, q_3, is possible even after it has been expanded. This prompts us to check for potential update of all the neighbors of an expanding vertex rather than only the open neighbors (line 8 of Algorithm 1). Such reopening of expanded vertices are however possible only if there exists non-acute simplices in the complex, as will be proven in Section 4.2. (a) When q_2 is expanded, and the $\overline{d}$-value of q_3 is updated, the simplex $\{q_2, q_3, q_4\}$ has not been generated yet since, in addition to q_2 and q_3, there is another open vertex, q_4, that constitutes the simplex. (b) Next, when q_3 is expanded (before q_4, since it has a lower $\overline{d}$-value), potential updates are made to the $\overline{d}$-value of q_4 and other neighbors of q_3, but not q_3 itself. Simplex $\{q_2, q_3, q_4\}$ is generated, but only checked for path to q_4 (a neighbor of expanding vertex, q_3) through that simplex. (c) However, when q_4 is being expanded, we can potentially improve the $\overline{d}$-value of its neighbor, q_3, for a path through simplex $\{q_2, q_3, q_4\}$, even though q_3 has previously been expanded. We check for improvement, and if the $\overline{d}$-value of a previously expanded vertex is updated, we insert it back into the set Q (a process called "*reopening*;" see line 16 of Algorithm 1).

not be sufficient to expand only the immediate neighbors of the goal since reopening can cascade down to predecessors across multiple generations. Under such circumstances it is necessary to terminate the algorithm after expansion of a certain number of generations of children after the expansion of the goal vertex (i.e., stop when the expanded vertex, q, is at a distance of N hops from q_{goal} in G, and q_{goal} has been expanded).

4. When vertex q is being expanded, in order to compute a candidate for updating the $\overline{d}$-value of a neighboring vertex u, we generate all maximal simplices (in $\mathcal{R}(G)$) consisting of vertices q, u, and only other expanded vertices (line 10, and procedure "*MaximalSimplices*" is described in more detail in Section 2.1).
5. We compute a potential $\overline{d}$-value, d', for updating u for paths through each of these maximal simplices attached to $\{u, q\}$, and choose the lower out of that as the candidate to test against for update. This is computed by the "*DistanceThroughSimplex*" procedure described in Section 2.2. This procedure also returns the pair of data, $(\sigma', \overline{w}')$, which represents a "*came-from point*" inside simplex σ'. The *cfp* is however not used in the Basic S* algorithm itself, but is used in the path reconstruction discussed later (Algorithm 4). This is in contrast to the corresponding step in a graph search algorithm, where the potential value to test against is simply the sum of the $\overline{d}$-value at q and the length/cost of the edge $\{q, u\}$.

The remainder of this section is dedicated to providing the details of the two main subprocedures used in the Basic S* algorithm: *MaximalSimplices* (Section 2.1) and *DistanceThroughSimplex* (Section 2.2).

2.1. Computing attached maximal simplices

The *MaximalSimplices* procedure in the Basic S* algorithm computes all the "maximal simplices" attached to a simplex. We formalize the definitions and provide couple of Lemmas that allow the efficient computation of attached maximal simplices, the complete pseudocode for which appears in Algorithm 2.

Definition 4 (Maximal simplices and maximal simplices attached to a simplex)**.**

(a) *Maximal simplex constructed out of a set of vertices.* A maximal simplex constructed out of a set of vertices, $S \subseteq C_0$, is a subset $\chi \subseteq S$, such that χ is a simplex in $\mathcal{R}(G)$, and χ is not a face of any higher-dimensional simplex constituting of the vertices from S. We refer to the set of maximal simplices created out of S as $\mathcal{M}(S) \subseteq C_*$. Formally, $\mathcal{M}(S) = \{\chi \mid \chi \subseteq S, \chi \in C_* \text{ and } \chi \cup a \notin C_* \; \forall a \in S - \chi\}$.

(b) *Neighbors of a simplex.* A vertex $a \in C_0$ is called a neighbor of a simplex $\sigma \in C_*$ if $\{a, b\} \in C_1$ for all $b \in \sigma$ (i.e., a is connected to every vertex in σ by a 1-simplex). Thus, if $\sigma \in C_n$, then $\{a\} \cup \sigma \in C_{n+1}$. The set of all neighbors of σ in G is $\mathcal{N}_G(\sigma) = \bigcap_{v \in \sigma} \mathcal{N}_G(v)$ (where $\mathcal{N}_G(v)$ is the set of neighbors of v in G, excluding v itself).

(c) *Attached maximal simplex constructed out of a set of neighbors.* If $S \subseteq \mathcal{N}_G(\sigma)$ is a set of neighbors of a simplex $\sigma \in C_*$, then the set of maximal simplices constructed out of S and attached to σ is the set $\mathcal{M}(S; \sigma) := \mathcal{M}(S \cup \sigma)$. See Figure 5.

Algorithm 2: Construct attached maximal simplices

$\mathcal{MS} = MaximalSimplices_{\mathcal{R}(G)}(S;\ \sigma,\ \nu = \emptyset)$
Inputs: a. Rips complex, $\mathcal{R}(G) = \{C_0, C_1, C_2, \ldots\}$, of graph $G = (V, E)$.
b. Simplex $\sigma \in C_*$.
c. A set of neighbors, $S \subseteq \mathcal{N}_G(\sigma)$.
d. Neighbor restriction set, ν (defaults to empty set; required only for recursive call).
Outputs: a. The set of maximal simplices constructed out of S and attached to σ.

```
1  if (|S| > 1)
2      for each (m ∈ S, n ∈ S, m ≠ n)
3          if {m, n} ∉ C_1
4              a := m, b := n   // a, b ∈ S such that they are not connected.
5              break
6  if a, b are undefined   // σ ∪ S is maximal.
7      μ := σ ∪ S
8      if N_G(μ) ∩ ν = ∅
9          MS := {μ}
10     else
11         MS := {}   // one of the vertices from ν is a neighbor of α.
12 else
13     MS_(i) := MaximalSimplices_R(G) (S ∩ N_G(a); σ ∪ {a}, ν)
14     MS_(ii) := MaximalSimplices_R(G) (S ∩ N_G(b); σ ∪ {b}, ν)
15     MS_(iii) := MaximalSimplices_R(G) (S − {a, b}; σ, ν ∪ {a, b})
16     MS := MS_(i) ∪ MS_(ii) ∪ MS_(iii)
17 return MS.
```

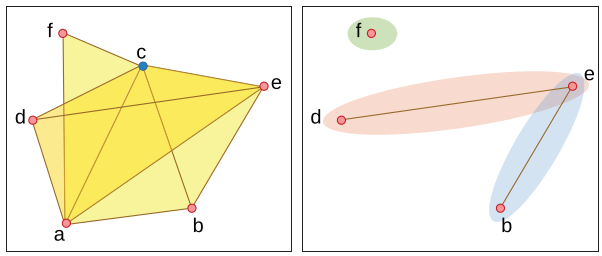
The following lemma simply states the fact that the union of a simplex, σ, with every maximal simplex in $\mathcal{M}(S)$, gives the set of maximal simplices constructed out of the set of vertices in $\sigma \cup S$.

Lemma 1. *If $\sigma \in C_*$ is a simplex, and S a set of neighbors of σ, (i.e., every vertex in S is connected to every vertex in σ), then $\mathcal{M}(\sigma \cup S) = \{\sigma \cup \chi \mid \chi \in \mathcal{M}(S)\}$.*

The direct way of computing $\mathcal{M}(S;\sigma)$ will be to check whether $\sigma \cup \alpha$ is a simplex in C_* for every $\alpha \in \mathcal{P}(S)$ (the *power set* of S). However the complexity of this algorithm would be $O(2^{|S|})$. The development of a more efficient algorithm for procedure *MaximalSimplices* (Algorithm 2) relies on the following observation.

Lemma 2. *Suppose $\sigma \in C_*$ and S is a set of neighbors of σ. Identify two vertices, $a, b \in S$ such that $\{a, b\} \notin C_1$, i.e., a and b not connected (if such a pair does not exist, then $\sigma \cup S$ is the only maximal simplex). Then the set $\mathcal{M}(S;\sigma)$ of maximal simplices constructed out of S and attached to σ can be partitioned into three parts.*

(i) Maximal simplices containing a, but not containing b: $\mathcal{M}(S \cap \mathcal{N}_G(a);\ \sigma \cup \{a\})$ (note: $S \cap \mathcal{N}_G(a)$ does not contain b, since a and b are not connected).
(ii) Maximal simplices containing b, but not containing a: $\mathcal{M}(S \cap \mathcal{N}_G(b);\ \sigma \cup \{b\})$.
(iii) Maximal simplices containing neither a nor b: $\{\gamma \in \mathcal{M}(S - \{a, b\}; \sigma) \mid a \notin \mathcal{N}_G(\gamma), b \notin \mathcal{N}_G(\gamma)\}$.



(a) (b)

Fig. 5. Maximal simplices attached to $\{\mathsf{a}, \mathsf{c}\}$ constructed out of its neighbors $\{\mathsf{b}, \mathsf{e}, \mathsf{d}, \mathsf{f}\}$. (a) In this subcomplex, the set of maximal simplices constructed out of the set $U = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}, \mathsf{f}\}$ is $\mathcal{M}(U) = \{\{\mathsf{a}, \mathsf{b}, \mathsf{e}, \mathsf{c}\}, \{\mathsf{a}, \mathsf{e}, \mathsf{c}, \mathsf{d}\}, \{\mathsf{a}, \mathsf{c}, \mathsf{f}\}\}$. This is also the set of maximal simplices attached to $\sigma = \{a, c\}$ and constructed out of $S = \{\mathsf{b}, \mathsf{e}, \mathsf{d}, \mathsf{f}\}$. Refer to Figure 2 for color index for the simplices. (b) The set $S = \{\mathsf{b}, \mathsf{e}, \mathsf{d}, \mathsf{f}\}$ is a set neighbors of the 1-simplex $\sigma = \{a, c\}$. Here $\mathcal{M}(S) = \{\{\mathsf{d}, \mathsf{e}\}, \{\mathsf{b}, \mathsf{e}\}, \{\mathsf{f}\}\}$. It is easy to observe that $\mathcal{M}(U) = \{\sigma \cup \chi \mid \chi \in \mathcal{M}(S)\}$. Furthermore, since d and b are not connected in the complex, $\mathcal{M}(S)$ can be partitioned into subset of simplices containing d but not b, containing b but not d, and containing neither.

Factoring in the computational overhead for the search of the pair $\{a, b\}$ and the computation in computing set intersections (which is an $O(k \log k)$ operation for sets of size k maintained using a heap, while hash table is expected linear time), the complexity of this algorithm is $O(|S|^3 + |S|^2 \log |S|) = O(|S|^3)$.

2.2. Distance of a vertex through a simplex

One of the key components of the Basic S* algorithm is the computation of the $\overline{d}$-distance of an *apex* of a simplex (the vertex that is potentially being updated) through one of the attached simplices (the "*DistanceThroughSimplex*" procedure). This is an extrapolation/estimation problem: *Given the $\overline{d}$-distance of the vertices that constitute the base of a simplex, and the lengths of the edges of the simplex, what is the $\overline{d}$-distance of the apex?* To answer this question we use the piece-wise linear approximation of the underlying configuration manifold that is represented by the simplicial complex, and this in turn lets us assume a flat (curvature-free) metric inside a simplex. We thus construct a canonical Euclidean embedding of the simplex, and that in turn lets us perform the extrapolation. With only local information available, we explore two possible extrapolation methods: *spherical* and *linear*.

We first provide a few preliminary definitions and propositions related to the canonical embedding of a metric simplex, followed by discussion on the said extrapolation methods, and finally a complete algorithm for the "*DistanceThroughSimplex*" procedure.

Definition 5 (A pointed simplex)**.** A *pointed simplex* is a simplex, σ, with a preferred vertex, $u \in \sigma$, called the *apex* of the simplex.

Without loss of generality, we refer to the vertices of a $(n-1)$-simplex, σ, as $v_0, v_1, v_2, \ldots, v_{n-1}$, with v_0 being the apex whenever σ is pointed, and arbitrarily chosen ordering for $v_1, v_2, \ldots, v_{n-1}$.

Definition 6 (A metric simplex)**.** A metric $(n-1)$-simplex is an $(n-1)$-simplex, σ, with a metric defined on the set σ, $d : \sigma \times \sigma \to \mathbb{R}_{\geq 0}$, satisfying all the axioms of a metric. With $\sigma = \{v_0, v_1, v_2, \ldots, v_{n-1}\}$, for brevity we will write $d(v_i, v_j) = d_{i,j} = d_{j,i}$ for all $v_i, v_j \in \sigma$. Thus, a metric $(n-1)$-simplex is defined by the pair (σ, d).

Definition 7 (An Euclidean realizable metric simplex)**.** A metric $(n-1)$-simplex, (σ, d), is called Euclidean realizable if its constituent vertices can be isometrically embedded in an Euclidean space (i.e., the Euclidean distance between the embedded vertices are equal to the distances between the vertices in the metric simplex).

Given an $(n-1)$-dimensional *metric simplex*, the following proposition gives a recipe for isometrically embedding it in an $(n-1)$-dimensional Euclidean space.

Proposition 1 (Canonical Euclidean realization of a metric simplex)**.** *There is an unique embedding of an Euclidean realizable metric $(n-1)$-simplex, $(\sigma = \{v_0, v_1, \ldots, v_{n-1}\}, d)$, given by $e : \sigma \to \mathbb{R}^{n-1}$ such that the following hold.*

(i) The embedded point for the jth vertex has non-zero value for the first j coordinates, with the jth coordinate being non-negative, and zero for the rest. That is, $\mathbf{v}_j := e(v_j) = [\mathsf{v}_{j,0}, \mathsf{v}_{j,1}, \ldots, \mathsf{v}_{j,j-1}, 0, \ldots, 0]$, $\mathsf{v}_{j,j-1} \geq 0$, $j = 0, 1, 2, \ldots, n-1$.
(ii) We have $\|\mathbf{v}_i - \mathbf{v}_j\| = d(v_i, v_j) = d_{i,j}$.

Explicitly, the embedding can be written using the following recursive formula:

$$\mathsf{v}_{j,k} = \begin{cases} \frac{1}{2\mathsf{v}_{k+1,k}} \Big(d_{j,0}^2 - d_{j,k+1}^2 + \mathsf{v}_{k+1,k}^2 \\ \qquad + \sum_{p=0}^{k-1} (\mathsf{v}_{k+1,p}^2 - 2\mathsf{v}_{j,p}\mathsf{v}_{k+1,p}) \Big), & k < j-1, \\ \sqrt{d_{j,0}^2 - \sum_{p=0}^{j-2} \mathsf{v}_{j,p}^2}, & k = j-1. \\ 0, & k \geq j. \end{cases} \tag{1}$$

where $\sum_{p=\alpha}^{\beta} h(p) = 0$ whenever $\beta < \alpha$. Using (1), the computation of $\mathsf{v}_{1,0}, \mathsf{v}_{2,0}, \mathsf{v}_{2,1}, \mathsf{v}_{3,0}, \mathsf{v}_{3,1}, \mathsf{v}_{3,2}, \ldots, \mathsf{v}_{j,0}, \mathsf{v}_{j,1}, \ldots, \mathsf{v}_{j,j-1}, \mathsf{v}_{j+1,0}, \ldots$ can be made in an incremental manner, with the computation of a term in this sequence requiring only the previous terms.

The proof of this proposition is constructive, and the construction appears in Appendix A.1. An illustration of this embedding of a simple 2-simplex is shown in Figures 3(d) and 6.

Definition 8 (Canonical Euclidean realization of metric simplex)**.** The map e described in Proposition 1 is referred to as the *canonical Euclidean realization* of the metric simplex (σ, d), and will be referred to as $\mathcal{E}_d(\sigma) : \sigma \to \mathbb{R}^{n-1}, v_i \mapsto \mathbf{v}_i$.

The above-described Euclidean realization of a metric simplex, if it exists, gives an isometric embedding of the simplex in $\mathbb{R}^{n-1}$. A lack of existence (e.g., when any of the $\mathsf{v}_{j,k}$ is complex) points to the fact that the simplex is not isometrically embeddable in an Euclidean space.

In Sections 2.2.1 and 2.2.2 we discuss the two extrapolation methods for computing the $\overline{d}$-distance of the apex, v_0, of a metric simplex given the $\overline{d}$-distances of the other vertices, $v_1, v_2, \ldots, v_{n-1}$. A complete algorithm for computing the extrapolated $\overline{d}$-distance is discussed in Section 2.2.3.

2.2.1. Spherical extrapolation for computing unrestricted $\overline{d}$-distance of an apex. The *spherical extrapolation* method assumes that the $\overline{d}$-distance to the vertices in the Euclidean realization of a simplex increases in a spherically symmetric manner (Figure 6). Computing the center of this sphere, $\mathbf{o} \in \mathbb{R}^{n-1}$, is the first step in this extrapolation method, and is described by the following proposition.

Proposition 2. *Given the canonical Euclidean realization, $\mathcal{E}_d(\sigma) = e : v_j \mapsto \mathbf{v}_j = [\mathsf{v}_{j,0}, \mathsf{v}_{j,1}, \ldots, \mathsf{v}_{j,j-1}, 0, \ldots, 0]$, of a pointed Euclidean realizable metric simplex, (σ, d) (with apex v_0), and given a map $\overline{d} : \{v_1, v_2, \ldots, v_{n-1}\} \to \mathbb{R}_+$, one can compute a point $\mathbf{o} = [\mathsf{o}_0, \mathsf{o}_1, \ldots, \mathsf{o}_{n-2}] \in \mathbb{R}^{n-1}$, using*

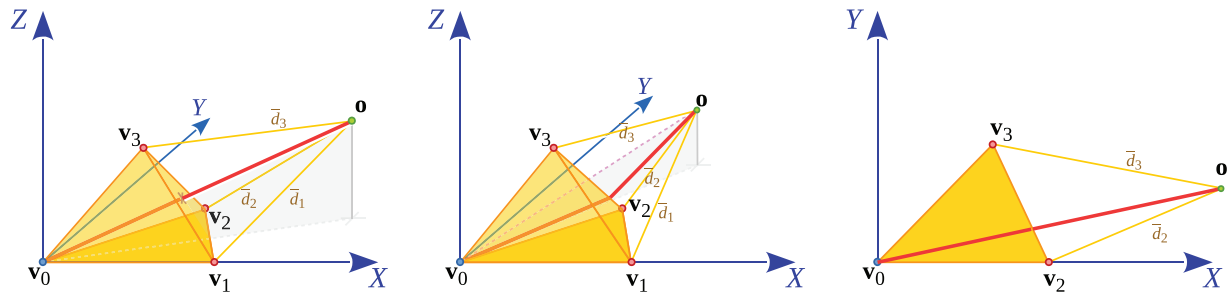


Fig. 6. Spherical extrapolation: Euclidean realization of a metric 3-simplices, $(\sigma = \{v_0, v_1, v_2, v_3\},\ d)$, and given two sets of values for $\bar{d} : \sigma - \{v_0\} \to \mathbb{R}_+$. (a) The $\bar{d}$-distance of an apex, v_0, through the simplex, $\sigma = \{v_0, v_1, v_2, v_3\}$, is the same as the Euclidean distance between $\mathbf{o}$ and $\mathbf{v}_0$ in the vertices' Euclidean embedding, since $\overline{W}^{sph}_{(d,\bar{d})}(\sigma, v_0) \geq \mathbf{0}$. (b) The line joining v_0 and $\mathbf{o}$ intersect H_0 outside the triangle formed by $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$. Not all elements of $\overline{W}^{sph}_{(d,\bar{d})}(\sigma, v_0)$ are non-negative. Thus, the $\bar{d}$-distance through simplex is not equal to the unrestricted $\bar{d}$-distance. (c) The $\bar{d}$-distance for (b) is computed by computing the unrestricted $\bar{d}$-distance through the face opposite to $\mathbf{v}_1$. This is computed by a completely separate Euclidean realization of the metric simplex $(\{v_0, v_2, v_3\}, d)$. Here $\overline{\mathbf{ov}_0}$ passes through the face made up of $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ in (a), but not in (b). Recursive computation of unrestricted $\bar{d}$-distances trough faces, (c), is required for computed the $\bar{d}$-distance for (b).

the following formula:

$$\begin{aligned} \mathsf{o}_0 &= \frac{-V + \sqrt{V^2 - 4UW}}{2U} \\ \mathsf{o}_k &= A_k \mathsf{o}_0 + B_k, \qquad k = 1, 2, \ldots, n-2 \end{aligned} \tag{2}$$

where

$$\begin{aligned} A_1 &= \tfrac{\mathsf{v}_{1,0} - \mathsf{v}_{2,0}}{\mathsf{v}_{2,1}} \\ A_k &= \tfrac{\mathsf{v}_{1,0} - \mathsf{v}_{k+1,0}}{\mathsf{v}_{k+1,k}} - \textstyle\sum_{p=1}^{k-1} \tfrac{\mathsf{v}_{k+1,p}}{\mathsf{v}_{k+1,k}} A_p, \quad k \geq 2 \\ B_1 &= \tfrac{1}{2\mathsf{v}_{2,1}} \left(\mathsf{v}_{2,0}^2 + \mathsf{v}_{2,1}^2 - \mathsf{v}_{1,0}^2 + \bar{d}_1^2 - \bar{d}_2^2 \right) \\ B_k &= \tfrac{\left(\sum_{p=0}^{k} \mathsf{v}_{k+1,p}^2 - \mathsf{v}_{1,0}^2 + \bar{d}_1^2 - \bar{d}_{k+1}^2 \right)}{2\mathsf{v}_{k+1,k}} - \textstyle\sum_{p=1}^{k-1} \tfrac{\mathsf{v}_{k+1,p}}{\mathsf{v}_{k+1,k}} B_p, \quad k \geq 2 \\ U &= \left(1 + \textstyle\sum_{p=1}^{n-2} A_p^2 \right) \\ V &= 2 \left(-\mathsf{v}_{1,0} + \textstyle\sum_{p=1}^{n-2} A_p B_p \right) \\ W &= \left(\mathsf{v}_{1,0}^2 - \bar{d}_1^2 + \textstyle\sum_{p=1}^{n-2} B_p^2 \right) \\ \bar{d}_j &:= d(v_j) \end{aligned}$$

A real solution to (2) *exists if and only if a point* $\mathbf{o}$ *exists in the same Euclidean space as the embedded metric simplex satisfying* $\|\mathbf{o} - \mathbf{v}_j\| = \bar{d}_j := \bar{d}(v_j)$. *In that case* $\mathbf{v}_0$ *and* $\mathbf{o}$ *are points lying on or on the opposite sides of the hyperplane containing* $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{n-1}$.

The proof, once again, is constructive, and appears in Appendix A.2.

Given an Euclidean realizable metric simplex, (σ, d), with apex v_0, we can construct its canonical Euclidean realization, $e := \mathcal{E}_d(\sigma)$, using (1). In addition, given the map $\bar{d}$, we can compute the coordinate of $\mathbf{o}$ using the above proposition. This lets us compute the extrapolated $\bar{d}$-distance of the apex, v_0, as follows.

Definition 9 (Unrestricted $\bar{d}$-distance of an apex)**.** Given the canonical Euclidean realization, $\mathcal{E}_d(\sigma) = e : v_j \mapsto \mathbf{v}_j$, of a pointed Euclidean realizable metric $(n-1)$-simplex, (σ, d), with apex v_0, we compute the point $\mathbf{o} \in \mathbb{R}^{n-1}$ satisfying the given distances $\|\mathbf{o} - e(v_j)\| = \bar{d}_j$, $j = 1, 2, \ldots, n-1$ using (2). We thus define the unrestricted $\bar{d}$-distance of v_0 to be $\overline{D}^{sph}_{(d,\bar{d})}(\sigma, v_0) = \|\mathbf{o} - e(v_0)\|$.

Here $\overline{D}^{sph}_{(d,\bar{d})}(\sigma, v_0)$ is the length of the line segment connecting $\mathbf{v}_0$ and $\mathbf{o}$ in the Euclidean realization (Figure 6). This line, $\overline{\mathbf{ov}_0}$, intersects the hyperplane, H_0, containing $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{n-1}\}$ at a general point that can be written as $\sum_{i=1}^{n-1} w_i \mathbf{v}_i$, where $\sum_{i=1}^{n-1} w_i = 1$. The following is a simple geometric consequence, and a derivation appears in Appendix A.3.

Proposition 3. *The point at which the line connecting* $\mathbf{v}_0$ *and* $\mathbf{o}$ *intersects the hyperplane* H_0 *is given by* $\mathbf{i}_0 = \sum_{i=1}^{n-1} w_i \mathbf{v}_i$, *with*

$$w_k = \frac{w'_k}{\sum_{i=1}^{n-1} w'_i}, \qquad k = 1, 2, \ldots, n-1 \tag{3}$$

where w'_j *can be computed recursively using the formula* $w'_j = \frac{\mathsf{o}_{j-1} - \sum_{i=j+1}^{n-1} w'_i \mathsf{v}_{i,j-1}}{\mathsf{v}_{j,j-1}}$. *Note that the terms in the sequence* $w'_{n-1}, w'_{n-1}, \ldots, w'_1$ *can be computed in an incremental manner.*

If all the w_j, $j = 1, 2, \ldots, n-1$ *are non-negative, then the line intersects the hyperplane inside (or on the boundary of) the Euclidean realization of the face of the simplex containing* $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{n-1}$. *Otherwise, it intersects outside.*

Definition 10 (Intersection point in spherical extrapolation)**.** For the weights computed using (3), we introduce the map $\overline{W}^{sph}_{(d,\bar{d})}(\sigma, v_0) : \sigma - \{v_0\} \to \mathbb{R},\ v_i \mapsto w_i,\ i = 1, 2, \ldots, n-1$.

The above method of computing the unrestricted $\overline{d}$-distance of the apex and the weights, w_j, relies on the construction of the point $\mathbf{o}$, and identifying the $\overline{d}$-distances as the distances from that point. Since we consider distance from a single point, $\mathbf{o}$, we refer to this method of computation "*spherical*." In the following we introduce an alternative to this computation.

2.2.2. Linear extrapolation for computing the unrestricted $\overline{d}$-distance of an apex. Instead of computing a point, $\mathbf{o} \in \mathbb{R}^{n-1}$, from which the distances of the points $\mathbf{v}_j$ are asserted to be $\overline{d}_j$, one can compute a $(n-2)$-dimensional hyperplane, I, in the Euclidean space in which the simplex is realized, from which the distances of the points $\mathbf{v}_j$ are assumed to be $\overline{d}_j$. This gives us an alternative extrapolation method for computing the $\overline{d}$-distance of the apex, v_0 (Figure 7). The following proposition summarizes the computation of the said plane, the proof of which appears in Appendix A.4.

Proposition 4. *Given the canonical Euclidean realization, $\mathcal{E}_d(\sigma) = e : v_j \mapsto \mathbf{v}_j = [\mathsf{v}_{j,0}, \mathsf{v}_{j,1}, \ldots, \mathsf{v}_{j,j-1}, 0, \ldots, 0]$, of a pointed Euclidean realizable metric simplex, (σ, d) (with apex v_0), and given a map $\overline{d} : \{v_1, v_2, \ldots, v_{n-1}\} \to \mathbb{R}_+$, one can compute a hyperplane, I, described by the equation $\mathbf{u} \cdot \mathbf{x} + \mu = 0$, where, $\mathbf{x} \in \mathbb{R}^{n-1}$ is a point on the hyperplane, $\mathbf{u} = [\mathsf{u}_0, \mathsf{u}_1, \ldots, \mathsf{u}_{n-2}] \in \mathbb{R}^{n-1}$ is an unit vector orthogonal to the plane, and μ is a constant using the following formulas:*

$$\mu = \frac{-Q + \sqrt{Q^2 - 4PR}}{2P},$$
$$\mathsf{u}_k = M_k \mu + N_k, \quad k = 0, 1, 2, \ldots, n-2 \tag{4}$$

where

$$\begin{aligned}
&M_0 = -1\\
&M_k = -\left(1 + \tfrac{1}{\mathsf{v}_{k+1,k}} \textstyle\sum_{p=0}^{k-1} M_p \mathsf{v}_{k+1,p}\right), \quad k \geq 1\\
&N_0 = \overline{d}_1\\
&N_k = \left(\overline{d}_{k+1} - \tfrac{1}{\mathsf{v}_{k+1,k}} \textstyle\sum_{p=0}^{k-1} N_p \mathsf{v}_{k+1,p}\right), \quad k \geq 1\\
&P = \textstyle\sum_{j=0}^{n-1} M_j^2\\
&Q = 2\textstyle\sum_{j=0}^{n-1} M_j N_j\\
&R = \textstyle\sum_{j=0}^{n-1} N_j^2 - 1\\
&\overline{d}_j := d(v_j)
\end{aligned}$$

A real solution to (4) *exists if and only if a plane I exists in the same Euclidean space as the embedded metric simplex such that the distances of the points $\mathbf{v}_j$ from the plane $I : \mathbf{u} \cdot \mathbf{x} + \mu = 0$ are $\overline{d}_j$ for $j = 1, 2, \ldots, n-1$. In that case the projection of the simplex constituting of points $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{n-1}$ on to the hyperplane I, and the point $\mathbf{v}_0 = \mathbf{0}$ lie on the opposite sides of the hyperplane containing $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{n-1}$.*

Definition 11 (Unrestricted $\overline{d}$-distance of an apex)**.** Given the canonical Euclidean realization, $\mathcal{E}_d(\sigma) = e : v_j \mapsto \mathbf{v}_j$, of a pointed Euclidean realizable metric $(n-1)$-simplex, (σ, d), with apex v_0, we compute the plane, I, described by the equation $\mathbf{u} \cdot \mathbf{x} + \mu = 0$ (with $\mathbf{u}$ an unit vector) satisfying $\mathbf{u} \cdot \mathbf{v}_j + \mu = \overline{d}_j$, $j = 1, 2, \ldots, n-1$, using (4). We thus define the unrestricted $\overline{d}$-distance of v_0 to be $\overline{D}^{lin}_{(d,\overline{d})}(\sigma, v_0) = \mathbf{u} \cdot \mathbf{v}_0 + \mu$.

Proposition 5. *The point at which the perpendicular dropped from $\mathbf{v}_0$ to the plane, I, intersects the hyperplane H_0 is given by $\mathbf{i}_0 = \sum_{i=1}^{n-1} w_i \mathbf{v}_i$, with*

$$w_k = \frac{{w'}_k}{\sum_{i=1}^{n-1} {w'}_i}, \quad k = 1, 2, \ldots, n-1 \tag{5}$$

where ${w'}_j$ can be computed recursively using the formula ${w'}_j = \frac{\mathsf{u}_{j-1} - \sum_{i=j+1}^{n-1} {w'}_i \mathsf{v}_{i,j-1}}{\mathsf{v}_{j,j-1}}$. Note that the terms in the sequence ${w'}_{n-1}, {w'}_{n-1}, \ldots, {w'}_1$ can be computed in an incremental manner.

If all the w_j, $j = 1, 2, \ldots, n-1$, are non-negative, then the line intersects the hyperplane inside (or on the boundary of) the Euclidean realization of the face of the simplex containing $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{n-1}$. Otherwise, it intersects outside.

Definition 12 (Intersection point in linear extrapolation)**.** For the weights computed using (5), we introduce the map $\overline{W}^{lin}_{(d,\overline{d})}(\sigma, v_0) : \sigma - \{v_0\} \to \mathbb{R}$, $v_i \mapsto w_i$, $i = 1, 2, \ldots, n-1$.

A feature of the linear method is that with the $\overline{d}$-distances being distances from a plane, there is no notion of triangle inequality or other properties of a metric that can be defined on $\overline{d}$. This may or may not be desirable under different situations. Given a fine-enough discretization, both the spherical and linear approaches should approximate the underlying metric relatively well. In our implementation we use only the spherical extrapolation.

2.2.3. Algorithm for computing the $\overline{d}$-distance through simplex. As discussed, if some of the weights given by $\overline{W}^{*}_{(d,\overline{d})}(\sigma, v_0)$ (where the "$*$" refers to either "*sph*" or "*lin*" depending on the chosen extrapolation method) are negative, then the line of the shortest length from the apex to $\mathbf{o}$ (in spherical extrapolation) or I (in linear extrapolation) will not pass through the face simplex constituting of $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{n-1}$. In that case we need to compute the length of the shortest path that intersects H_0 inside the simplex constituting of $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{n-1}$. Such a path, clearly, will pass through one of the faces opposite to $\mathbf{v}_1, \mathbf{v}_2, \ldots$ or $\mathbf{v}_{n-1}$ and is the $\overline{d}$-distance of v_0 restricted to the face (see Figure 6(b) or 7(b)).

Likewise, it may be possible that the weights, $\overline{W}^{*}_{(d,\overline{d})}(\sigma, v_0)$, are complex because the components of $\mathbf{o}$ (for the spherical extrapolation) or the vector $\mathbf{u}$ (for linear

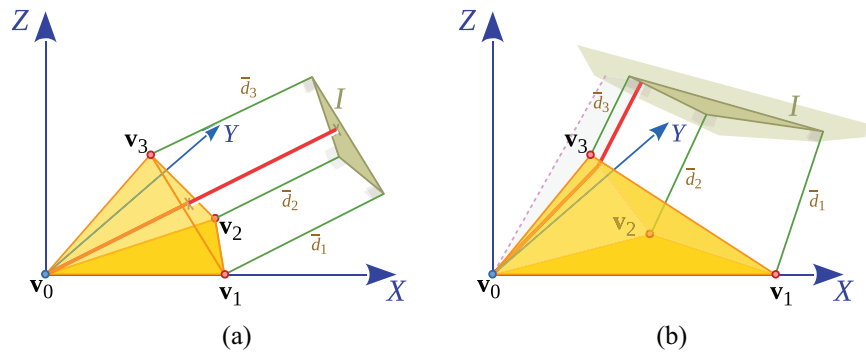


Fig. 7. Linear extrapolation: Euclidean realization of a metric 3-simplice, (σ, d), and given two sets of values for $\bar{d} : \sigma - \{v_0\} \to \mathbb{R}_+$. (a) The $\bar{d}$-distance of an apex, v_0, through the simplex, $\sigma = \{v_0, v_1, v_2, v_3\}$, is the same as the Euclidean distance between $\mathbf{v}_0$ and the plane I in the vertices' Euclidean embedding, since $\overline{W}^{lin}_{(d,\bar{d})}(\sigma, v_0) \geq \mathbf{0}$. (b) The perpendicular dropped from v_0 to I intersect H_0 outside the triangle formed by $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$. Not all elements of $\overline{W}^{lin}_{(d,\bar{d})}(\sigma, v_0)$ are non-negative. Thus, the $\bar{d}$-distance through simplex is not equal to the unrestricted $\bar{d}$-distance. The perpendicular dropped from $\mathbf{v}_0$ to I passes through the face made up of $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ in (a), but not in (b).

extrapolation) are complex (see the formulas in Propositions 2 and 4). These can happen when the metric in the graph is too skewed (for example, when the lengths of the edges constituting a simplex violate triangle inequality). Under such a situation, we once again consider each of the faces of the simplex, and re-attempt the computation of the weights in the one lower-dimensional Euclidean space corresponding to the face simplex.

This recursion through the faces of a simplex continues until the algorithm hits the dimension of one, in which case it is guaranteed to find a solution since the embedding of $\mathbf{o}$ or computation of $\mathbf{u}$ in a one-dimensional Euclidean space is trivial and guaranteed to exist ($\mathbf{o}$ would be a point on the line containing $\mathbf{v}_1$ and $\mathbf{v}_2$, and at a distance $\bar{d}_1$ from $\mathbf{v}_1$, on the opposite side of $\mathbf{v}_0$). Thus, in that case the particular simplex is reduced to a graph and the algorithm behaves as Dijkstra's when computing the distance to $\mathbf{v}_0$ for that simplex. Thus, our algorithm for computing $\bar{d}$-distance through the simplex is one that computes and compares the $\bar{d}$-distance of v_0 restricted to the faces until a set of real, non-negative weights are found. The complete pseudocode is given in the following.

Note that the procedure also return the pair $(\sigma', \overline{w}')$, where σ' is a simplex (consisting of vertices with non-zero weights) and $\overline{w}' : \sigma' \to (0, 1]$ is a map that associates weights to the vertices in the simplex. These two pieces of information, $(\sigma', \overline{w}')$, together describes a point inside the simplex σ' as the convex combination $\sum_{v \in \sigma'}^{m} \overline{w}(v)\ v$.

During the recursive calls to *DistanceThroughSimplex*, it is possible that the procedure is called on the same simplex multiple times (by different faces). In order to avoid re-computation of the $\bar{d}$-value through the same simplex multiple times, we maintain a lookup table of the simplices (as a hash table maintained globally across all calls to *DistanceThroughSimplex*), and return the *DistanceThroughSimplex* value if it exists. The entries in the table (the *hash key*) are distinguished by the simplex vertices (unordered), the simplex's apex as well as the $\bar{d}$-value of the other vertices.

3. Path reconstruction

We now present a path reconstruction algorithm from the $\bar{d}$ field computed by *Basic S** to compute explicit shortest paths in the complex. An abstract point inside a simplicial complex can be described by two pieces of information: (i) an m-simplex, $\rho = \{v_0, v_1, \ldots, v_m\} \in C_*$, inside (or on the boundary of) which the point lies, and (ii) a set of positive weights associated with each vertex such that they add up to 1 (we represent the weights by the map $\overline{w} : \sigma \to \mathbb{R}_{\geq 0}$, with $\sum_{v \in \rho} \overline{w}(v) = \sum_{i=0}^{m} \overline{w}(v_i) = 1$). The point itself would then be the abstract convex combination $\sum_{v \in \rho} \overline{w}(v) = \sum_{i=0}^{m} \overline{w}(v_i)\ v_i$. The pair $p = (\sigma, \overline{w})$ described a point, and a (piece-wise linear) path in a simplicial complex can thus be described as a sequence of points $p_i = (\sigma_i, \overline{w}_i)$, $i = 0, 1, 2, \ldots$.

The primary output of the Basic S* algorithm (Algorithm 1) is a $\bar{d}$-value for every vertex in G. In order to find the shortest path connecting $s \in V$ with some arbitrary $g \in V$ (which has been expanded), like any search algorithm, we need to reconstruct a path. The basic reconstruction algorithm is described in Algorithm 4.

The procedure *ComputeCameFromPoint* takes in a point, $p_i = (\sigma_i, \overline{w}_i)$, on a simplex σ_i, and returns another point, $p_{i+1} = (\sigma_{i+1}, \overline{w}_{i+1})$ on a different simplex that it *came from* (Figure 8). As a result, the relationship between the two simplices, σ_i and σ_{i+1} is that they both are subsimplices of a same *maximal simplex*. Thus, given p_i, in order to compute p_{i+1}, we construct all the maximal simplices attached to σ_i (Line 4 of Algorithm 5), interpret the point p_i as a vertex in each of the maximal simplices (Lines 8, 9), and construct pointed metric simplices using faces of each of them and p_i as apex. The complete pseudocode is given in Algorithm 5.

Algorithm 3: Compute distance through simplex

$\big(d', (\sigma', \overline{w}')\big) = DistanceThroughSimplex_{(d,\,\overline{d})}\,(\sigma, u)$
Inputs: a. The metric simplex, σ, with distance between vertices $d : \sigma \times \sigma \to \mathbb{R}_+$.
b. Apex, $u \in \sigma$. (Thus, letting $\sigma = \{v_0 := u, v_1, v_2, \ldots, v_{n-1}\}$.)
c. $\overline{d}$-values, $\overline{d} : \sigma - \{u\} \to \mathbb{R}_+$.
Outputs: a. $\overline{d}$-distance to u through simplex σ.
b. Came-from point, $(\sigma', \overline{w}')$, on a subsimplex $\sigma' \subseteq \sigma - \{u\}$.

If (σ, u) exists in lookup table with the specified $\overline{d}$-values for $\sigma - \{u\}$,
  return its distance-through-simplex value and came-from point.
$\overline{w} := \overline{W}^*_{(d,\overline{d})}(\sigma, u)$ // weights associated with $v_i, i = 1, 2 \cdots, n-1$
if $\overline{w}(v_i)$ is real AND $\overline{w}(v_i) \geq 0,\ \forall i = 1, 2, \ldots, n-1$ // all weights are real non-negative
  $d' := \overline{D}^*_{(d,\overline{d})}(\sigma, u)$.
  $\sigma' := \{v \in \sigma - \{u\} \mid \overline{w}(u) > 0\}$ // simplex constituting of vertices with non-zero weights
  $\overline{w}'(v) := \overline{w}(v),\ \forall v \in \sigma'$ // weights describing *came-form* point in simplex σ'
else
  $d' := \infty$
  foreach $(i \in \{1, 2, \ldots, n-1\})$
    if $(\overline{w}(v_i)$ is complex OR $\overline{w}(v_i) < 0)$ // Need to check $\overline{d}$-value through face.
      $\big(d'', (\sigma'', \overline{w}'')\big) = DistanceThroughSimplex_{(d,\overline{d})}\,(\sigma - v_i,\ u)$ // distance through face opposite to v_i.
      Insert an entry for $(\sigma - v_i, u)$ with the specified $\overline{d}$-values for $\sigma - \{v_i, u\}$ into lookup table,
        with d' its distance-through-simplex value and $(\sigma'', \overline{w}'')$ its came-from point.
      if $d'' < d'$
        $d' \leftarrow d''$
        $(\sigma', \overline{w}') \leftarrow (\sigma'', \overline{w}'')$ // came-from point.
return $d', (\sigma', \overline{w}')$

(where the "$*$" in $\overline{W}^*$ and $\overline{D}^*$ refers to either "*sph*" or "*lin*" depending on the chosen extrapolation method.)

Algorithm 4: Reconstruct path

$\{p_i\}_{i=0,1,2,\ldots} = ReconstructPath_{(\mathcal{R}(G),\,s,\,d,\overline{d})}\,(g)$
Inputs: a. The Rips complex of graph, $G = (V, E)$.
b. Length/cost of the edges, $d : V \to \mathbb{R}_+$.
c. The $\overline{d}$-values of the vertices (obtained as output of S* algorithm).
d. The vertex, g, to find the shortest path from.
Outputs: a. A sequences of points, $p_i = (\sigma_i, \overline{w}_i)$ for $i = 0, 1, 2, \ldots$.

$\overline{w}_0(g) := 1$ // weight map, $\overline{w} : \{g\} \to (0, 1],\ g \mapsto 1$
$p_0 := (\{g\}, \overline{w}_0)$ // 0-simplex consisting only of g, with a weight of 1 associated with it.
$\mu := \emptyset$
$i := 0$
while $(p_i \neq (\{s\},\ \overline{w} : \{s\} \to (0, 1], s \mapsto 1))$
  $(p_{i+1}, \mu') := ComputeCameFromPoint_{(\mathcal{R}(G),\,d,\,\overline{d})}(p_i, \mu)$
  $\mu \leftarrow \mu'$
  $i \leftarrow i + 1$
return $\{p_i\}_{i=0,1,2,\ldots}$

Algorithm 5: Compute came-from point in path reconstruction

$\left((\sigma', \overline{w}'), \mu'\right) = ComputeCameFromPoint_{(\mathcal{R}(G),s,d,\overline{d})}((\sigma, \overline{w}), \mu)$

Inputs: a. The Rips complex of graph, $G = (V, E)$.
b. Length/cost of the edges, $d : V \to \mathbb{R}_+$.
c. The $\overline{d}$-values of the vertices (obtained as output of S* algorithm).
d. A point, $p = (\sigma, \overline{w})$, in the complex.
e. Maximal simplex, μ, to be ignored.

Outputs: a. A point, $p' = (\sigma', \overline{w}')$, in the complex.
b. The maximal simplex, μ', through which the line $\overline{p_i p_{i+1}}$ passes.

1 $d^{ex} := d$ // extended distance map.
2 $S := \mathcal{N}(\sigma)$ // common neighbors of all the vertices in σ.
3 $d' := \infty$ // distance to p.
4 $\mathcal{MS} = MaximalSimplices_{\mathcal{R}(G)}(S; \sigma)$ // maximal simplices attached to σ
5 **foreach** ($\rho \in \mathcal{MS} - \{\mu\}$) // all attached maximal simplices, except μ.
6 $\overline{w}^{ex} := \overline{w}$; $\overline{w}^{ex}(v) := 0, \forall v \in \rho - \sigma$ // extend domain of $\overline{w}$ to the simplex ρ.
7 $e := \mathcal{E}_d(\rho)$ // canonical Euclidean embedding of ρ
8 $\mathbf{p} := \sum_{v \in \rho} \overline{w}^{ex}(v)\ e(v)$ // coordinates of $p = (\sigma, \overline{w})$ in canonical embedding of ρ.
9 $d^{ex}(v, p) = d^{ex}(p, v) := \|e(v) - \mathbf{p}\|, \forall v \in \rho$ // extend domain of distance/cost function to include $\{v, p\}$.
10 **foreach** ($u \in \rho$)
11 $\gamma := \rho - \{u\}$ // face of ρ opposite to u
12 **if** ($\gamma \not\subseteq \mu$) // ignore any subsimplex of μ.
13 $\left(d'', (\sigma'', \overline{w}'')\right) := DistanceThroughSimplex_{(d^{ex}, \overline{d}^{ex})}(\gamma \cup \{p\}, p)$
// $\overline{d}$-distance of p through simplex $\gamma \cup \{p\}$ with p as apex.
14 **if** ($d'' < d'$)
15 $d' \leftarrow d''$
16 $(\sigma', \overline{w}') \leftarrow (\sigma'', \overline{w}'')$
17 $\mu' \leftarrow \rho$
18 **return** $\left((\sigma', \overline{w}'), \mu'\right)$

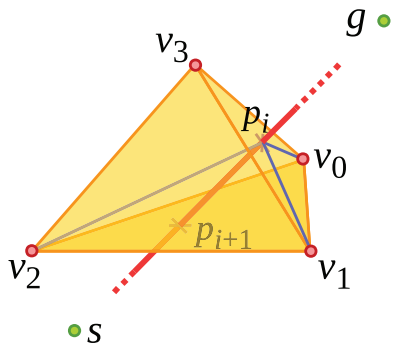


Fig. 8. Came-from point: the point p_i is given by the pair $(\sigma = \{v_1, v_0, v_3\}, \overline{w})$. Here $\rho = \{v_0, v_1, v_2, v_3\}$ is a maximal simplex attached to σ. Its face, $\gamma = \{v_0, v_1, v_2\}$, contains the came-from point of p_i. To determine the $\overline{d}$-value of p, we need to construct the simplex $\gamma \cup \{p\}$, and compute the distance to apex p through that simplex.

4. Analysis

4.1. Correctness

We construct maximal simplices attached to every edge, $\{u, q\}$, in line 10 of the Basic S* algorithm (Algorithm 1) and use it for computing and testing for update of the $\overline{d}$-value of a vertex u that is a neighbor to the expanding vertex, q. Since the edge itself is a sub-1-simplex of each of those maximal simplices, the distance, d', through the maximal simplex (line 12) cannot be greater than the distance obtained by simply adding $d(\{u, q\})$ to $\overline{d}(q)$ (consequence of triangle inequality). As a result, we have the following proposition comparing the cost of paths computed by Basic S* with that of paths computed by graph search algorithms in which the solution path is restricted to the graph (for example, A* and Dijkstra's)..

Proposition 6 ($\overline{d}$-value bounded above by graph search value)**.** *The Basic S* algorithm (Algorithm 1) cannot return a higher value for the distance to vertex ($\overline{d}$-values) than one obtained using a graph search algorithm in which paths are restricted to the graph.*

Furthermore, a simplicial complex with constant metric (metric with zero curvature) inside each simplex gives a piece-wise linear approximation of any Riemannian manifold (Jost, 2008). The canonical Euclidean embedding of the simplices indeed induces such constant metrics on the simplices (any constant Riemannian metric can be converted to Euclidean metric through appropriate scalings (Petrunin, 2003)). Thus, the proposed method is appropriate for computing shortest paths in discrete representations of

Riemannian manifolds. As a direct consequence, we have the following proposition.

Proposition 7 ($\overline{d}$-value approaches path metric on a Riemannian manifold)**.** *Suppose M is a Riemannian manifold (possibly with boundaries), and $\mathcal{R}(G)$ is a simplicial approximation of the manifold (with the cost of the edges in G set to their lengths on the manifold). Then the cost/distance between two points on the manifold computed using the Basic S* algorithm converges to the actual distance between the same points on the manifold as the size of the simplices (lengths/costs of edges in G) approach zero.*

A formal proof of the above statement appears in Appendix A.5, where we also show that the error in the computed $\overline{d}$-distance from the true path metric is at the worst $O(\epsilon^2)$, where ϵ is the average size of the simplices.

4.2. No reopening in complex with acute simplices

Consider the shortest path to a vertex, v, computed during the Basic S* algorithm. A simplex that contains the last segment of the path to v will be referred to as a *came-through* simplex of v. Thus, for example, $\{u, q_1, q_2\}$ is a came-through simplex of u in Figure 3d, whereas $\{q_2, q_3, q_4\}$ is a came-through simplex of q_3 in Figure 4.

Lemma 3. *If a vertex, v, is reopened during the Basic S* search, then there exists a came-through simplex of v in which v is not the vertex with highest $\overline{d}$-value.*

Proof. Suppose v is reopened during the expansion of vertex q. We consider the simplex consisting of v, q, and the *came-from* simplex in $cfp(v)$. This is clearly a came-through simplex of v. However, the facts that q was expanded after v and that the reopening further reduced the $\overline{d}$-value of v, mean that the $\overline{d}$-value of q is higher than the $\overline{d}$-value of u. □

Proposition 8. *A sufficient condition for vertices to be not reopened during the Basic S* search is that every simplex in $\mathcal{R}(G)$ be* acute*.*

Proof. We prove this proposition assuming the *linear extrapolation* model. A proof using the *spherical extrapolation* is similar.

Consider an Euclidean realization of an n-dimensional metric simplex, σ, with the coordinates of the embedded vertices $\mathbf{v}_j \in \mathbb{R}^n, j = 0, 1, 2, \ldots, n$. Since we consider the linear extrapolation model, iso-$\overline{d}$ surfaces in this embedding will be hyperplanes. Consider the $(n-1)$-dimensional hyperplanes of uniform $\overline{d}$-value, $H_j = \{\mathbf{u} \in \mathbb{R}^n \mid \overline{d}(\mathbf{u}) = \overline{d}(\mathbf{v}_j)\}$. In addition, let the ray pointing along the perpendicular to H_j passing through $\mathbf{v}_j$ in the direction of decreasing $\overline{d}$-value (which is also the ray along the negative gradient of $\overline{d}$-value, and hence the direction in which the shortest path to $\mathbf{v}_j$ points; see Bhattacharya et al. (2014) for a proof) be $H_j^{\perp} = \{\mathbf{u} \in \mathbb{R}^n \mid (\mathbf{u} - \mathbf{v}_j) \cdot (\mathbf{v} - \mathbf{v}_j) = 0 \ \forall \mathbf{v} \in H_j,\ \overline{d}(\mathbf{u}) \leq \overline{d}(\mathbf{v}_j)\}$ (see Figure 9).

Since the angle between H_j and $H_j^{\perp}$ is a right angle, if the simplex is acute, for every vertex $\mathbf{v}_j$, at most one out of H_j and $H_j^{\perp}$ can intersect the simplex non-trivially (at points other than $\mathbf{v}_j$). In general, there are exactly two vertices at which H_j do not intersect the simplex: the vertices with the maximum and minimum $\overline{d}$-values. Out of them only at the vertex at which $\overline{d}$-values is maximum the ray $H_j^{\perp}$ *can* potentially intersect the simplex non-trivially. At all other vertices, since H_j intersects the simplex non-trivially, $H_j^{\perp}$ cannot intersect the simplex non-trivially.

In other words, the only vertex of the acute simplex that can potentially have the simplex as a *came-through* simplex is the vertex with the highest $\overline{d}$-value. This, combined with Lemma 3 proves the statement of the proposition. □

4.3. Complexity

The Basic S* algorithm (Algorithm 1) has an overall structure very similar to Dijkstra's algorithm. If the graph, G, has $|V|$ counts of vertices that are expanded, and assuming that no vertex (or a finite number of vertices) is reopened, then the main while block of the algorithm (starting at line 5) will loop for $|V|$ times. Inside each loop, the following processes happen.

(i) The vertex with lowest $\overline{d}$-value is extracted from set Q (line 6). The size of Q (open list) is of the order of a constant power of the size of V, and since Q is maintained in a heap data structure, the complexity of this step is $O(\log |Q|) = O(\log |V|^k) = O(\log |V|)$.

(ii) We loop through each neighbor of each vertex to check for updates (line 8). Assuming average degree of each vertex is D, this loops for $O(D)$ times. Inside each of these subloops, the following computations happen.
   (a) The algorithm for computing the set of maximal simplices attached to each edge (line 10), as discussed in Section 2.1, is of complexity $O(|S|^3) = O(D^3)$ (where S is the set of neighbors of an edge).
   (b) The duality between maximal simplices and the vertices tells us that the average number of maximal simplices attached to each edge is also D. Thus, the innermost loop (starting at line 11 that includes the "*DistanceThroughSimplex*" call) loops for $O(D)$ times.

Thus, the overall complexity of the algorithm is $O\left(|V|\left(\log |V| + D^4 + D^2\right)\right) = O\left(|V|\left(\log |V| + D^4\right)\right)$.

In a simplicial complex with only acute simplices, Basic S* does not reopen any vertex. However in complexes with obtuse simplices we need to account for the additional time complexity owing to vertex reopening. A vertex can be independently reopened only if a new attached simplex with better path-through-simplex cost is created (an

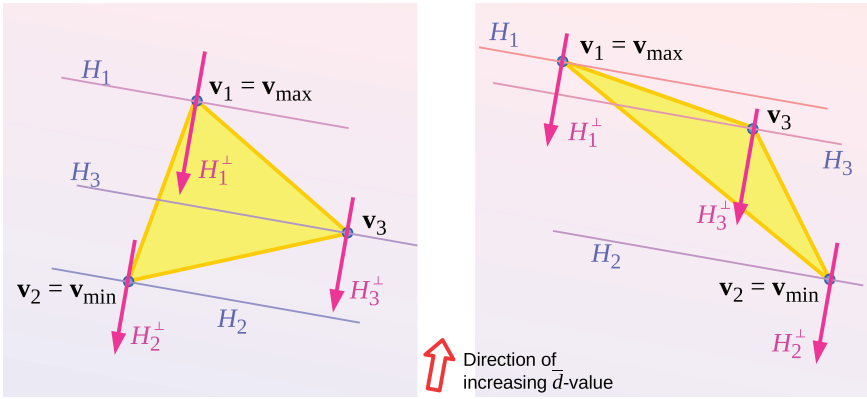


Fig. 9. An acute simplex (*left*) versus an obtuse simplex (*right*). Since H_j intersect the simplex for every vertex that is not a vertex with maximum/minimum $\bar{d}$-value, the only way $H_j^\perp$ can also intersect the simplex at a vertex other than the maximum/minimum $\bar{d}$-value vertices is when a dihedral angle of the simplex at the vertex is obtuse (for example, vertex $\mathbf{v}_3$ in the right figure).

$O(D)$ number of such attached simplex would exist for each vertex). Re-opening can also happen if a successor of a vertex is reopened and the effect cascades down to itself. Thus, in the worst case, reopening of a vertex can cascade down to every other previously expanded vertices. This means that in the worst case for every vertex in V, all the previously expanded vertices will be re-expanded, $O(|D|)$ times, and hence there will be $O(D|V|^2)$ total expansions. In that case the overall complexity of the algorithm will be $O\left(D|V|^2\left(\log|V| + D^4\right)\right)$. This, however, is extremely unlikely and would require every simplice in the complex to be obtuse.

4.3.1. Complexity as size of configuration space increases keeping dimension constant. If the average degree of the vertices is finite and constant, and if $|V| \to \infty$, then the complexity is simply $O(|V|\log|V|)$ assuming no or finite number of reopenings (for example, when all but a finite number of simplices in the complex are acute), and is $O\left(|V|^2\log|V|\right)$ in the worst case when every vertex is reopened and the reopening cascades down to all its predecessors.

4.3.2. Complexity as dimension of configuration space increases keeping diameter constant. If N is the dimension of the configuration space, and the *diameter* of the configuration space is held constant, then as $N \to \infty$ we have $|V| \to O(e^N), D \to O(e^N)$. Thus, assuming finite number of reopening, the complexity of the algorithm as $N \to \infty$ is $O\left(e^N\left(\log(e^N) + e^{4N}\right)\right) = O(e^{5N})$. In the worst case when every vertex is reopened and the reopening cascades down to all the predecessors of every vertex, the complexity is $O\left(e^N e^{2N}\left(\log(e^N) + e^{4N}\right)\right) = O(e^{7N})$. In contrast, algorithms such as Dijkstra's, that can plan paths in discrete graph representation of configuration manifolds of arbitrary dimensions, the time complexity grows as $O(e^N)$ with the dimension of the configuration manifold, which, although exponential, is lower than Basic S*.

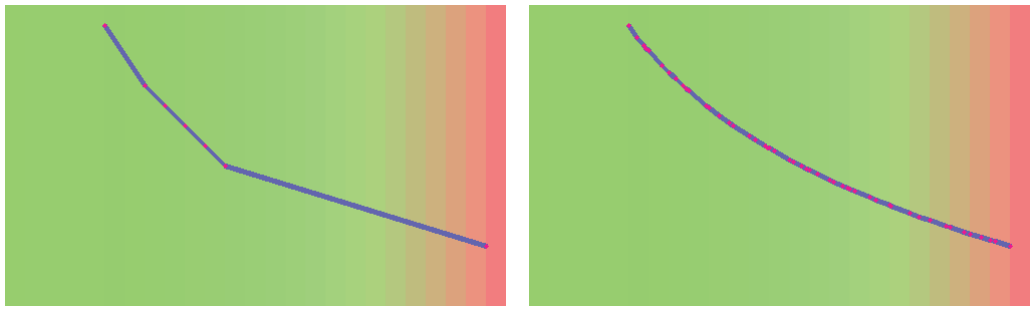
Fig. 10. An example path (in magenta) on the game map, "AR0071SR," computed using Basic S* (experiment # 2).

5. Results

We implemented the Basic S* algorithm in C++. Our current implementation, although not fully optimized for performance, is available at https://github.com/subh83/DOSL. In this section we present results demonstrating several applications of the Basic S* algorithm along with comparison with other search algorithms.

R1. *Simple demonstration*: As a very simple demonstration, we present a comparison between the progress of search in a graph constructed out of a uniform triangulation (using equilateral triangles) of an Euclidean plane with a single obstacle (Figure 12). Figure 12(k) illustrates how refinement of the discretization around obstacle corners return lower cost paths.

R2. *Validation on a game map*: We tested our algorithm on a map (Figure 10) from Nathan Sturtevant's *pathfinding benchmarks* repository (Sturtevant, 2012) (game map



(a) (b)

Fig. 11. In this simple non-uniform quadratic traversal cost scenario (*green*: lower cost, *red*: higher cost), while Basic S* marginally outperforms Theta* in terms of the cost of the path computed (about 0.7%), qualitatively the path obtained from Basic S* is much smoother than that obtained from Theta*. (a) Path computed using Theta* has a total cost of 50.335547. (b) Path computed using Basic S* has a total cost of 49.973305.

"AR0071SR" from the *Baldurs Gate II*, 512 × 512 collection). We compared the performance of our algorithm against ANYA (Harabor and Grastien, 2013), which is guaranteed to be optimal on subsets of the Euclidean plane. The average suboptimality in our path length over the 1,279 experiments provided with the map was 0.013%, which is comparable with the best state-of-the-art any-angle path planning algorithm on Euclidean subsets of a plane (Uras and Koenig, 2015) (for example, on this same class of maps, the average suboptimality of Theta* and L-Theta* is of the order of 0.04%, whereas suboptimality of Field A* and Block A* is of the order of 0.3%, as has been presented in Uras and Koenig (2015)).

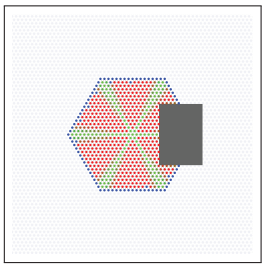
R3. *Comparison with Theta* for non-uniform traversal cost*: Theta* is a simple and powerful algorithm for searching in graphs with given Euclidean embeddings, and can handle non-uniform traversal costs. We implemented Theta* for non-uniform traversal cost as described in Daniel et al. (2010) and compared its performance against Basic S* in an obstacle-free 25 × 15 map discretized into a square grid (8-connected), with the simple quadratic traversal cost function $c(x,y) = 1.0 + 0.02x^2$. The results are shown in Figure 11. As can be observed, in this example the cost of the path computed using Theta* is 0.7% higher than the cost of the path computed using S* algorithm (which, although not negligible, is not significantly higher). However, qualitatively the path obtained from Basic S* is much smoother since the end-points of the line segments constituting the path in S* are not restricted to lie at the vertices of the graph, as is the case with Theta*.

It should also be noted that Basic S* requires only local connectivity information of the graph – for an expanding vertex, it needs to know what are its immediate neighbors in the graph, and the cost of the edges connecting to them, whereas Theta* requires us to know a priori what large curves/segments connecting distant vertices in the graph are (segments connecting a vertex and its far grandparent) to be able to check its feasibility and compute its cost.

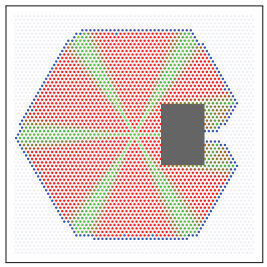
R4. *Shortest path on a 2-sphere*: We next present the result of computing the shortest path (geodesic) on the surface of a unit sphere. We use the usual spherical coordinates, (ϕ,θ), where $\phi \in [0,\pi]$ is the latitudinal angle measured from the positive Z axis, and $\theta \in [0,2\pi)$ is the longitudinal angle measured from the positive X axis (Figure 13(a)).

The matrix representation of the Riemannian metric tensor (Jost, 2008) in this coordinate system is $g = \begin{bmatrix} 1 & 0 \\ 0 & \sin^2\phi \end{bmatrix}$ (the *round metric*). Thus, an infinitesimal segment at (ϕ,θ) of extent $\Delta\phi$ along the ϕ direction and $\Delta\theta$ along the θ direction will be of length/cost $\Delta\ell = \sqrt{\Delta\phi^2 + \sin^2\phi\,\Delta\theta^2}$. In particular, if two closely lying points, v_1 and v_2, have spherical coordinates (ϕ_1,θ_1) and (ϕ_2,θ_2), respectively, we compute the cost/length of the line segment connecting those points as $d(v_1,v_2) = \sqrt{(\phi_2-\phi_1)^2 + \sin^2\left(\frac{\phi_1+\phi_2}{2}\right)(\theta_2-\theta_1)^2}$. For any longer curves or line segments in the chart, we subdivide it into small line segments and add up their individual costs/lengths to compute the total cost/length of the curve or line segment.

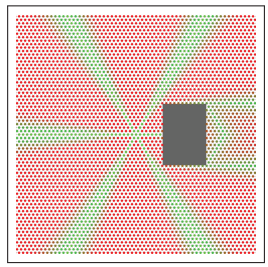
We construct the graph, G, out of a vertex set that has vertices placed on an uniform square lattice, with the separation of the neighboring vertices in the ϕ or θ direction being equal to π/f, where f is the "*fineness*" of the discretization (larger the value of f, finer is the discretization). Figure 13(b) shows such a discretization of the chart with $f = 8$. On a 2-sphere the shortest paths are the geodesics or the great circles, which can be computed analytically between two points on the sphere. The Riemannian metric on the latitude–longitude coordinate chart however induces a non-uniform (and anisotropic) traversal cost on the coordinate chart on which search algorithms can be employed. Comparison of paths computed using Dijkstra's search (red curves), Theta* search for non-uniform traversal cost (magenta curves), and the Basic S* search (blue



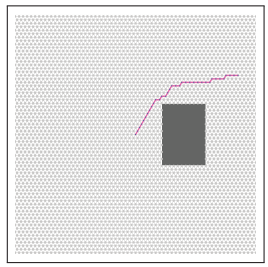
(a) Dijkstra's search: 600 vertices expanded.



(b) 2300 vertices expanded.

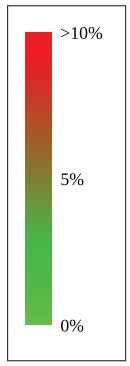


(c) all vertices in domain expanded.

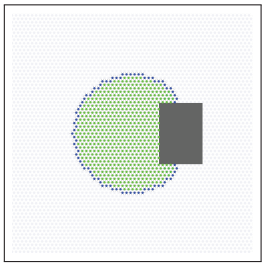


(d) Path connecting a vertex to the start.

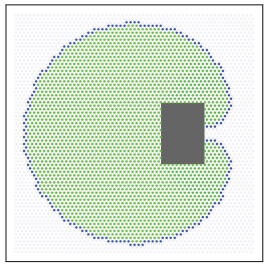
(a)–(d) Dijkstra's search algorithm.



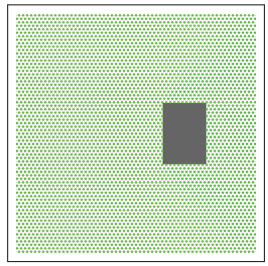
(i) Vertex color legend for the relative error between the distance computed by search algorithm and the true length of shortest path computed using Euclidean metric.



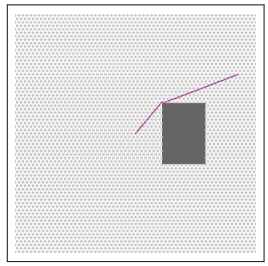
(e) Basic S* search: 600 vertices expanded.



(f) 2300 vertices expanded.

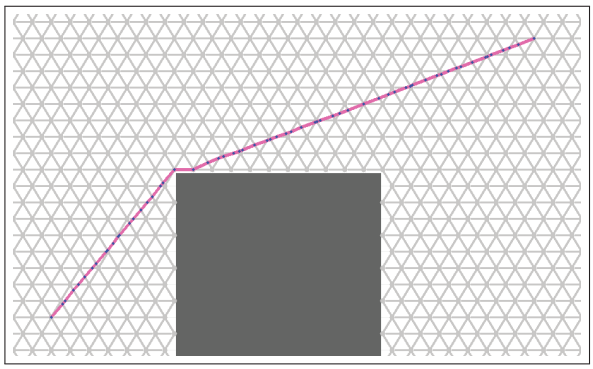


(g) all vertices in domain expanded.

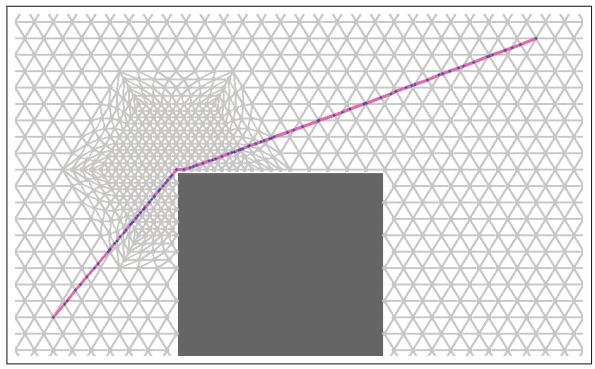


(h) Path connecting a vertex to the start.

(e)–(h) Basic S* search algorithm.



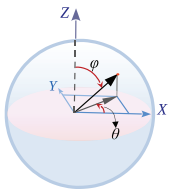
(j) Close-up view of the path computed in (h) using Basic S* search. Blue dots show the points $\{p_i\}_{i=0,1,\ldots}$ returned by the *ReconstructPath* procedure. Note how they lie on the boundaries of the maximal 2-simplices. Also note the obvious suboptimality near the corner of the obstacle where the path turns. Total cost of this path is 30.4417.



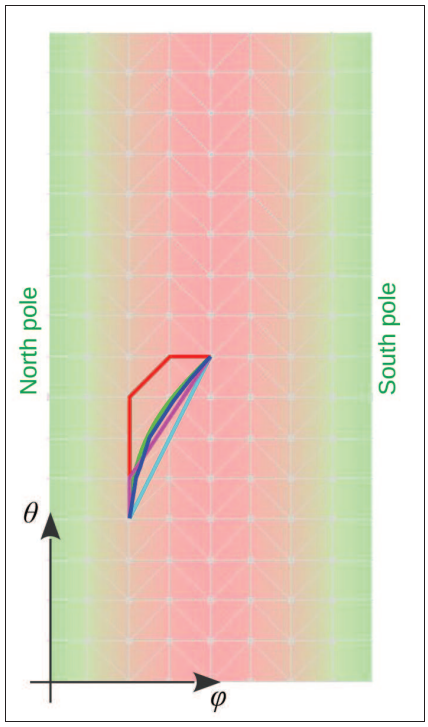
(k) A refinement in the discretization around the corner of the obstacle reduces the suboptimality (we used a simple inversion/reflection about a hexagon to create the refinement). The cost of this path is 30.4002. This demonstrates that refinements in the discretization can give better quality of solution using Basic S*.

Fig. 12. Comparison between Dijkstra's search and Basic S* search using a graph constructed out of an uniform triangulation of an Euclidean plane with a single obstacle. Expanded vertices colored in red and green, while vertices in the open list are in blue. *Top row:* Progress of Dijkstra's search in (a)–(c). *Second row:* Progress of Basic S* search in (e)–(g). Color of the vertices, (i), indicates the relative error between the distance to a vertex as computed by the search algorithm and the true length of shortest path (geodesic distance) to the point using the Euclidean metric on the plane (*green*: low error; *red*: high error). Note how the Basic S* search algorithm computes a distance that more closely represents the underlying path metric induced by the Euclidean metric.

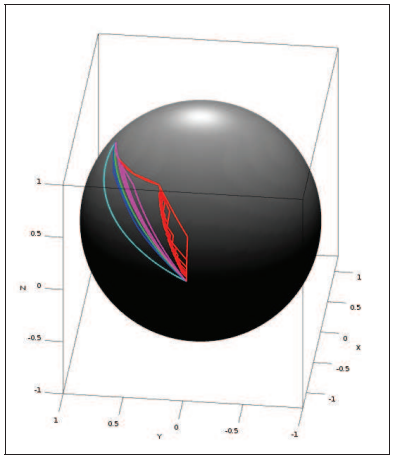
curves) are shown in Figure 13. As we increase the fineness value, f, it is to be noted that the graphs for lower fineness are not subsets of the graphs of higher fineness. Thus, interestingly, the cost of the paths computed using Dijkstra's search increase with fineness (Figure 13(d)), but the cost of the paths computed using Basic S* decreases and approaches the geodesic path (great circle) on the sphere. Theta*, on the other hand, performs fairly well in computing a path close to a geodesic path, although as the fineness is changed, in the worst case it returns a path that is about 1.3% more expensive than the path computed using Basic S*.



(a) The spherical coordinates.

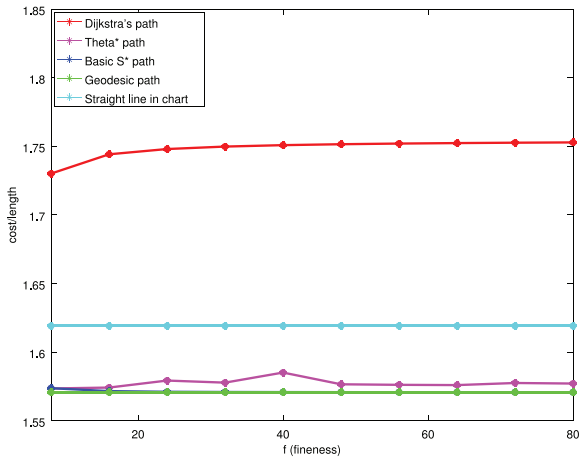


(b) The latitude–longitude, (ϕ, θ), coordinate chart. A triangulation graph (light gray) on the coordinate chart, with fineness, $f = 8$, is shown. The background color indicates the determinant of the metric tensor, which corresponds to *cost* of edges in a particular region (higher: red; lower: green).



(c) Paths connecting two points on the surface of a sphere. Red paths are computed using Dijkstra's, magenta paths are computed using Theta*, while blue paths are computed using Basic S*. Multiple paths of each of these colors correspond to different values of fineness, f, used in the discretization.

(b), (c) Paths computed on the surface of an unit sphere using Dijkstra's/A* search (red), Theta* (magenta), and Basic S* search (blue). For reference, the figure also shows the geodesic path (green) and the path corresponding to a straight line segment on the (ϕ, θ) coordinate chart (cyan). (b) shows the paths on the (ϕ, θ) coordinate chart computed for a discretization of fineness $f = 8$. (c) shows the paths on the surface of a sphere for various values of fineness, $f = 8, 16, 24, \ldots, 80$.



(d) The total cost/length of paths computed using Dijkstra's (red), Theta* (magenta) and Basic S* (blue) using different fineness values for the discretization (data points at $f = 8, 16, 24, \ldots, 80$). The cost of two reference paths (geodesic on sphere: green, straight line segment on coordinate chart: cyan) are also shown.

Fig. 13. Computation of shortest path connecting two points on a 2-sphere.

R5. ***Shortest path in three-dimensional Euclidean space with obstacles*****:** Figure 14 shows an example in three-dimensional Euclidean space with obstacles. For A*/Dijkstra's search, each vertex was connected to 26 neighbors. For basic S*, each vertex was connected to 14 neighbors as shown in the inset of Figure 14 (since more

connections than that will create redundant simplices). Path computed using Dijkstra's is about 8.6% more expensive than one computed using Basic S*.

R6. ***Path planning for a three-degree-of-freedom planar robot arm*****:** We present the result of motion planning for a three-degree-of-freedom planar robot arm with obstacles in its *workspace* (the two-dimensional plane in which the arm physically resides). For a robot arm such as this, the unconstrained joint configuration space is the 3-torus, $\mathbb{T}^3$, which cannot be embedded in the three-dimensional Euclidean space. A general unconstrained configuration of the arm is given by $(\theta_0, \theta_1, \theta_2)$, where θ_0 is the angle that the first segment makes with the positive X axis, while θ_i, $i = 1, 2$, is the angle made by the ith segment with the $(i-1)$th segment. A vertex, $(\theta_0, \theta_1, \theta_2)$, in the discrete graph description of the configuration space has neighbors of the form $(\theta_0 \pm \Delta_0, \theta_1 \pm \Delta_1, \theta_2 \pm \Delta_2) \mod 2\pi$, for fixed angle steps $\Delta_i = 2\pi/n_i$ (this gives the description of the *neighbor function*, $\mathcal{N}_G$).

In the Basic S* algorithm, as we expand vertices and generate neighbors of expanding vertices in the configuration space (using the neighbor function), we check for collision of the robot arm at the generated configuration at an on-demand basis, and add a configuration to the open list only if the corresponding robot arm is not intersecting an obstacle in the workspace. We thus do not explicitly construct the inadmissible regions/obstacles in the joint configuration space, $\mathbb{T}^3$, to begin with, since that is an expensive computation to perform. This example thus illustrates the local nature of Basic S*, wherein we can compute paths in configuration spaces that cannot be embedded in an Euclidean space, and the graph, G, is presented only in terms of local connectivity information.

For the cost/distance function, for neighboring vertices v_1 and v_2 in the configuration space separated by $\Delta\theta_i$ along the θ_i coordinate, we emulate the approximate energy consumption in moving the robot arm from one configuration to another using the following formula: $d(v_1, v_2) = \sqrt{\sum_{i=0}^{2} l_i^2 \Delta\theta_0^2 + \sum_{i=1}^{2} l_i^2 \Delta\theta_1^2 + l_2^2 \Delta\theta_2^2}$, where l_i is the length of the ith segment (with the segment connected to the fixed base indexed 0 and the one with the end-effector indexed 2). Note that this cost function penalizes the movement of the joint at the base of the arm (i.e., changing θ_0) more than any other joint, since turning the base joint implies that the whole arm would move, while turning one closer to the end-effector will imply that only a part of the arm will move. Figure 15 shows the motion computed using Basic S* for three consecutive planning problems for a robot arm. Note how the arm turns a full circle to reach back to the position where it started.

As far as we know, Basic S* is the only any-angle type algorithm that can be used in this robotic manipulator example: it is a three-dimensional configuration manifold, with non-uniform traversal cost, with non-trivial topology and no a priori (or well-defined) line-of-sight information in the configuration space.

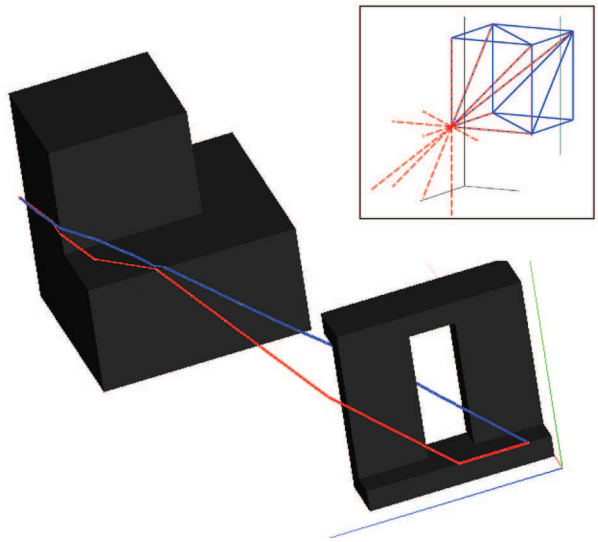


Fig. 14. Trajectories in a three-dimensional Euclidean space with obstacles. Blue: Computed using Basic S*. Red: Computed using Dijkstra's. The top right inset shows the connectivity of the vertices that result in a simplicial decomposition of the free space.

R7. ***Topological path planning using Basic S******:** Corresponding to the graph representation of the free configuration space, G, one can construct *homology*- or *homotopy*-augmented graph, G_h (Bhattacharya et al., 2015a,b, 2012), performing search in which one can compute trajectories in different topological classes. While in the past we have demonstrated topological path planning using Dijkstra's and S* search in finely discretized configuration graphs as well as visibility graphs, as a demonstration of an application we show an example of topological path planning using Basic S* algorithm. We use the Rips complex of the augmented graph, $\mathcal{R}(G_h)$, for optimal path planning in different topological classes using the Basic S* algorithm (Figure 16). Note that algorithms that require *line-of-sight* check cannot be used for topological path planning, since the line of sight is not a well-defined notion when the sought solution is restricted to a specific topological class.

R8. ***Runtime comparison*****:** As discussed previously, Basic S* needs to perform more computation per vertex than an algorithm such as Dijkstra's, even though the complexity of both the algorithms is $O(|V| \log |V|)$. As a result Basic S* runs slower when evaluated on the same environment using implementations with similar overheads across all the algorithms (available at https://github.com/subh83/DOSL). This is illustrated in the total computation time comparison presented in Figure 17.

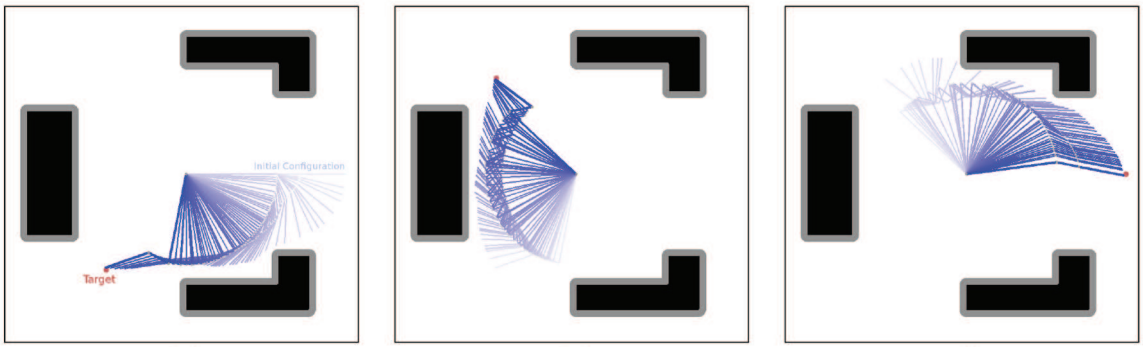


Fig. 15. Solutions of three consecutive path planning problems computed using Basic S* for a 3-degree of freedom planar robot arm with obstacles in its workspace. (a) Path planning problem no. 1. (b) Path planning problem no. 2. (c) Path planning problem no. 3. Obstacles are shown in black, with gray region around them showing "inflated" zone of safety. The target of each path planning problem is to make the end-effector reach the shown red disk. Each image shows the sequence of arm configurations that describe the path (the points p_i in the configuration space returned by *ComputeCameFromPoint*) computed using Basic S*: the final arm configuration is darker, while the initial configuration is lighter.

Fig. 16. Topological path planning: paths computed in three different homotopy classes in a game map, all using Basic S*. Red crosses show the *representative points* on the obstacles.

6. Discussion and conclusion

Computing an optimal path in a configuration space is fundamental to solving optimal motion planning problems in robotics, and hence an important problem in autonomy and artificial intelligence. Traditionally graphs have been the choice of discrete representation of configuration spaces for solving such problems, but graphs being one-dimensional entities fail to capture the richer topological and metric information of the underlying configuration space. In this paper we propose the use of simplicial complex as the more suitable choice of discrete representation, and hence present the *Basic S** algorithm for computing shortest paths through simplicial complexes. In particular, we consider the Rips complexes of graphs constructed as discrete representation of a configuration space of arbitrary topology, geometry, or dimension.

We also presented a path reconstruction algorithm for explicitly computing the shortest paths through the simplicial complex. The Basic S* algorithm has complexity comparable with Dijkstra's search, but is able to compute paths of significantly lower costs that closely match with the optimal paths (geodesics) in the original configuration space, as is demonstrated by our simulation results. We also presented comparison of cost of paths computed by an any-angle path planning algorithm. We provided a detailed complexity analysis. Our current implementation of the Basic S* algorithm is not fully optimized for a fair computation time comparison with other methods.

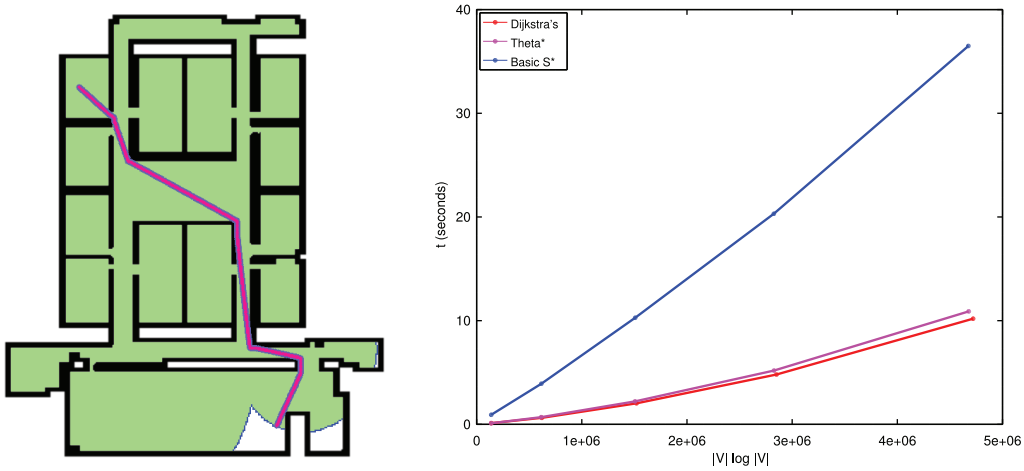
As part of the future work we will optimize the implementation of the algorithm and will incorporate heuristic functions into the search algorithm. Effective sampling techniques for constructing the graph (which we currently assume to be given to Basic S* in terms of local connectivity information), G, that capture small features of the configuration space (algorithms such as RRT and PRM) will be explored. We will also investigate the possibility of developing any-time/incremental computations and randomized searches within this framework. Distributed implementation of the algorithm will also be made.

Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Notes

1. A graph constitute of the vertex set, V, in which the elements are points (zero-dimensional entities), and the edge set, E, in which the elements are curves/line segments (one-dimensional entities). So the union of these sets has a dimension equal to the highest dimension of any element in the



(a) (b)

Fig. 17. Computation time increases almost linearly with $|V|\log|V|$ for Basic S* (*blue*), Dijkstra's (*red*), and Theta* (*magenta*). (a) Indoor office environment used for evaluation of computation time (also showing the shortest path computed using Basic S*). (b) Comparison of computation time of different algorithms as the number of vertices expanded is increased (owing to increasing size of the scaled environment). The slight nonlinearity appears because of additional overhead that the processor encounters in memory management owing to increasing memory consumption by the open list as the search size increases. Since Basic S* has to perform more computation per vertex, the overall time consumption is higher for Basic S* (almost four times that of Dijkstra's or Theta*).

union, which is 1 for a graph. This notion of dimension is consistent with definitions of dimension such as Hausdorff dimension, Minkowski dimension, and topological dimension. In particular, we do not refer to the dimension of the space in which a graph is embedded to be the dimension of the graph itself.

2. It is a simple fact that the coordinate representation of the gradient of the distance function, $\nabla\widetilde{d} = [\frac{\partial\widetilde{d}}{\partial x_1}, \frac{\partial\widetilde{d}}{\partial x_2}, \ldots, \frac{\partial\widetilde{d}}{\partial x_{n-1}}]$, is a vector pointing in the direction of the tangent to the geodesic when the matrix representation of the metric tensor is identity (see, for example, Bhattacharya et al., 2014). This, in fact, holds true even when the metric is only locally Riemannian in the neighborhood of the point at which the gradient is computed.

ORCID iD

Subhrajit Bhattacharya https://orcid.org/0000-0001-9139-054X

References

Bhattacharya S, Ghrist R and Kumar V (2014) Multi-robot coverage and exploration on Riemannian manifolds with boundary. *The International Journal of Robotics Research* 33(1): 113–137.

Bhattacharya S, Ghrist R and Kumar V (2015a) Persistent homology for path planning in uncertain environments. *IEEE Transactions on Robotics*. DOI: 10.1109/TRO.2015.2412051.

Bhattacharya S, Kim S, Heidarsson H, Sukhatme G and Kumar V (2015b) A topological approach to using cables to separate and manipulate sets of objects. *The International Journal of Robotics Research* 34(6): 799–815.

Bhattacharya S, Likhachev M and Kumar V (2010) Multi-agent path planning with multiple tasks and distance constraints. In: *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, Anchorage, AK.

Bhattacharya S, Likhachev M and Kumar V (2012) Topological constraints in search-based robot path planning. *Autonomous Robots* 33(3): 273–290.

Bondy J and Murty U (2007) *Graph Theory* (*Graduate Texts in Mathematics*). New York: Springer.

Burden RL and Faires JD (1989) *Numerical Analysis*. 4th Ed. Boston, MA: PWS Publishing Co.

Cohen B, Chitta S and Likhachev M (2010) Search-based planning for manipulation with motion primitives. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Conner DC, Rizzi A and Choset H (2003) Composition of local potential functions for global robot control and navigation. In: *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pp. 3546–3551.

Cormen TH, Leiserson CE, Rivest RL and Stein C (2009) *Introduction to Algorithms*. 3rd Ed. Cambridge, MA: MIT Press.

Cui M, Harabor DD and Grastien A (2017) Compromise-free pathfinding on a navigation mesh. In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI 2017)*, Melbourne, Australia, 19–25 August 2017, pp. 496–502.

Daniel K, Nash A, Koenig S and Felner A (2010) Theta*: Any-angle path planning on grids. *Journal of Artificial Intelligence Research* 39: 533–579.

Desai JP and Kumar V (1999) Motion planning for cooperating mobile manipulators. *Journal of Robotic Systems* 16: 557–579.

Dijkstra EW (1959) A note on two problems in connexion with graphs. *Numerische Mathematik* 1: 269–271.

Felner A, Zahavi U, Holte R, Schaeffer J, Sturtevant N and Zhang Z (2011) Inconsistent heuristics in theory and practice. *Artificial Intelligence* 175(9): 1570–1603.

Ferguson D, Baker CR, Likhachev M and Dolan JM (2008) A reasoning framework for autonomous urban driving. In: *Proceedings of the IEEE Intelligent Vehicles Symposium (IV 2008)*, Eindhoven, Netherlands, pp. 775–780.

Ferguson D and Stentz A (2007) Field d*: An interpolation-based path planner and replanner. In: *Robotics Research*. Berlin: Springer, pp. 239–253.

Hansen EA and Zhou R (2007) Anytime heuristic search. *Journal of Artificial Intelligence Research* 28: 267–297.

Harabor D and Grastien A (2013) An optimal any-angle pathfinding algorithm. In: *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 308–311.

Hart PE, Nilsson NJ and Raphael B (1968) A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science, and Cybernetics* 4(2): 100–107.

Hatcher A (2001) *Algebraic Topology*. Cambridge: Cambridge University Press.

Hong DW, Kimmel S, Boehling R, et al. (2009) Development of a semi-autonomous vehicle operable by the visually impaired. In: *2008 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pp. 455–467.

Hsieh MA, Loizou S and Kumar V (2007) Stabilization of multiple robots on stable orbits via local sensing. In: *IEEE International Conference on Robotics and Automation*, Rome, Italy.

Jiang K, Seneviratne LS and Earles SWE (1993) Finding the 3D shortest path with visibility graph and minimum potential energy. In: *Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems '93 (IROS '93)*, Vol. 1, pp. 679–684.

Jost J (2008) *Riemannian Geometry and Geometric Analysis*. Berlin: Springer.

Kavraki LE, Svestka P, Latombe JC and Overmars MH (1996) Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 12(4): 566–580.

Khosla P and Volpe R (1988) Superquadric artificial potentials for obstacle avoidance and approach. In: *Proceedings 1988 IEEE International Conference on Robotics and Automation*, Philadelphia, PA.

Kim JO and Khosla PK (1992) Real-time obstacle avoidance using harmonic potential functions. *IEEE Transactions on Robotics and Automation* 8(3): 338–349.

Kim S, Bhattacharya S and Kumar V (2014) Path planning for a tethered mobile robot. In: *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, China.

Kopczynski E, Pak I and Przytycki P (2012) Acute triangulations of polyhedra and $\mathbb{R}^n$. *Combinatorica* 32(1): 85–110.

L'Afflitto A and Sultan C (2010) Calculus of variations for guaranteed optimal path planning of aircraft formations. In: *2010 IEEE International Conference on Robotics and Automation*, pp. 1972–1977.

LaValle SM and Kuffner J (2001) Randomized kinodynamic planning. *The International Journal of Robotics Research* 20(5): 378–400.

Likhachev M and Stentz A (2008) R* search. In: *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.

Lozano-Pérez T and Wesley MA (1979) An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of ACM* 22(10): 560–570.

Mellinger D, Kushleyev A and Kumar V (2012) Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams. In: *Proceedings of the 2012 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 477–483.

Mitchell JSB, Mount DM and Papadimitriou CH (1987) The discrete geodesic problem. *SIAM Journal of Computing* 16(4): 647–668.

Montemerlo M, Becker J, Bhat S, (2008) Junior: The stanford entry in the urban challenge. *Journal of Field Robotics* 25(9): 569–597.

Nash A, Daniel K, Koenig S and Felner A (2007) Theta*: Any-angle path planning on grids. In: *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*. AAAI Press, pp. 1177–1183.

Nash A, Koenig S and Tovey CA (2010) Lazy theta*: Any-angle path planning and path length analysis in 3D. In: Fox M and Poole D (eds.) *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*. AAAI Press.

Petrunin A (2003) Polyhedral approximations of Riemannian manifolds. *Turkish Journal of Mathematics* 27(1): 173–188.

Rimon E and Koditschek D (1991) The construction of analytic diffeomorphisms for exact robot navigation on star worlds. *Transactions of the American Mathematical Society* 327(1): 71–116.

Stentz A (1995) The focussed D* algorithm for real-time replanning. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1652–1659.

Sturtevant N (2012) Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games* 4(2): 144–148.

Swaminathan S, Phillips M and Likhachev M (2015) Planning for multi-agent teams with leader switching. In: *Proceedings of ICRA*. IEEE, pp. 5403–5410.

Uras T and Koenig S (2015) An empirical comparison of any-angle path-planning algorithms. In: *Proceedings of the Eighth Annual Symposium on Combinatorial Search (SOCS)*, Dead Sea, Israel, pp. 206–211.

Uras T, Koenig S and Hernández C (2013) Subgoal graphs for optimal pathfinding in eight-neighbor grids. In: *Proceedings of the International Conference on Automated Planning and Scheduling*.

Urmson C, Anhalt J, Bae H, et al. (2008) Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics Special Issue on the 2007 DARPA Urban Challenge, Part I* 25(1): 425–466.

Xu C, Wang TY, Liu YJ, Liu L and He Y (2015) Fast wavefront propagation (FWP) for computing exact geodesic distances on meshes. *IEEE Transactions on Visualization and Computer Graphics* 21(7): 822–834.

Yershov DS and LaValle SM (2011) Simplicial Dijkstra and A* algorithms for optimal feedback planning. In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3862–3867.

Zefran M (1996) *Continuous Methods for Motion Planning*. PhD Thesis, University of Pennsylvania, Philadelphia, PA, USA.

Appendix A. Euclidean realization of metric simplices

Given a metric $(n-1)$-simplex, (σ, d), one can construct an isometric embedding of the simplex in the Euclidean $(n-1)$-space, $e : \sigma \to \mathbb{R}^{n-1}$ such that $\|e(v_i) - e(v_j)\| = d_{ij}$ and $\mathbf{v}_i := e(v_i)$ has all its last $n-1-i$ coordinates set to zero. The explicit construction can be described as follows.

Suppose $\mathbf{v}_j \in \mathbb{R}^{n-1}$ such that

$$\begin{aligned}
\mathbf{v}_0 &= [\ \ 0, \quad 0, \quad \cdots \quad 0\ \ ] \\
\mathbf{v}_1 &= [\ \ \mathsf{v}_{1,0}, \quad 0, \quad \cdots \quad 0\ \ ] \\
\mathbf{v}_2 &= [\ \ \mathsf{v}_{2,0}, \quad \mathsf{v}_{2,1}, \quad \cdots \quad 0\ \ ] \\
&\ \ \vdots \\
\mathbf{v}_j &= [\ \ \mathsf{v}_{j,0}, \quad \mathsf{v}_{j,1}, \quad \cdots, \ \mathsf{v}_{j,j-1}, \quad 0, \quad \cdots \quad 0\ \ ] \\
&\ \ \vdots \\
\mathbf{v}_{n-1} &= [\ \mathsf{v}_{n-1,0}, \ \mathsf{v}_{n-1,1}, \ \cdots, \ \mathsf{v}_{n-1,j-1}, \ \mathsf{v}_{n-1,j}, \ \cdots \mathsf{v}_{n-1,n-2}]
\end{aligned}$$

From the above, one obtains for $0 \le l < j \le n-1$,

$$\begin{aligned}
& d_{j,l}^2 \\
&= \|\mathbf{v}_j - \mathbf{v}_l\|^2 \\
&= \begin{cases} \sum_{p=0}^{j-1} \mathsf{v}_{j,p}^2, & l = 0 \\ \sum_{p=0}^{l-1} (\mathsf{v}_{j,p} - \mathsf{v}_{l,p})^2 + \sum_{p=l}^{j-1} \mathsf{v}_{j,p}^2, & 0 < l < j \le n-1 \end{cases}
\end{aligned} \tag{6}$$

A.1. Recursive formula for computing $\mathsf{v}_{j,k}$

Using (6) we compute $\mathsf{v}_{j,k}$ for $0 \le k < j$, $0 < j$ as follows.

For $k = 0$, $j = 1$ clearly,

$$\mathsf{v}_{1,0} = \pm d_{1,0} \tag{7}$$

For $k = 0,\ 1 < j \le n-1$,

$$\begin{aligned}
d_{j,1}^2 - d_{j,0}^2 &= (\mathsf{v}_{j,0} - \mathsf{v}_{1,0})^2 + \textstyle\sum_{p=1}^{j-1} \mathsf{v}_{j,p}^2 - \sum_{p=0}^{j-1} \mathsf{v}_{j,p}^2 \\
&= \mathsf{v}_{1,0}^2 - 2\mathsf{v}_{j,0}\mathsf{v}_{1,0} \\
\Rightarrow \quad \mathsf{v}_{j,0} &= (d_{j,0}^2 - d_{j,1}^2 + \mathsf{v}_{1,0}^2)/2\mathsf{v}_{1,0}
\end{aligned} \tag{8}$$

For $0 < k < j-1,\ 1 < j \le n-1$,

$$\begin{aligned}
& d_{j,k+1}^2 - d_{j,0}^2 \\
&= \textstyle\sum_{p=0}^{k} (\mathsf{v}_{j,p} - \mathsf{v}_{k+1,p})^2 + \sum_{p=k+1}^{j-1} \mathsf{v}_{j,p}^2 - \sum_{p=0}^{j-1} \mathsf{v}_{j,p}^2 \\
&= \textstyle\sum_{p=0}^{k-1} (\mathsf{v}_{k+1,p}^2 - 2\mathsf{v}_{j,p}\mathsf{v}_{k+1,p}) + (\mathsf{v}_{k+1,k}^2 - 2\mathsf{v}_{j,k}\mathsf{v}_{k+1,k}) \\
\Rightarrow \quad \mathsf{v}_{j,k} &= \Big(d_{j,0}^2 - d_{j,k+1}^2 + \mathsf{v}_{k+1,k}^2 \\
&\qquad + \textstyle\sum_{p=0}^{k-1} (\mathsf{v}_{k+1,p}^2 - 2\mathsf{v}_{j,p}\mathsf{v}_{k+1,p}) \Big) / 2\mathsf{v}_{k+1,k}
\end{aligned} \tag{9}$$

For $k = j-1,\ 1 < j \le n-1$,

$$\begin{aligned}
d_{j,0}^2 &= \textstyle\sum_{p=0}^{j-1} \mathsf{v}_{j,p}^2 \\
\Rightarrow \quad \mathsf{v}_{j,j-1} &= \pm \sqrt{d_{j,0}^2 - \textstyle\sum_{p=0}^{j-2} \mathsf{v}_{j,p}^2}
\end{aligned} \tag{10}$$

The important property of (7), (8), (9), and (10) is that the computation of a term in the sequence $\mathsf{v}_{1,0},\ \mathsf{v}_{2,0}, \mathsf{v}_{2,1},\ \mathsf{v}_{3,0}, \mathsf{v}_{3,1}, \mathsf{v}_{3,2}, \ldots, \mathsf{v}_{j,0}, \mathsf{v}_{j,1}, \ldots, \mathsf{v}_{j,j-1},\ \mathsf{v}_{j+1,0}, \ldots, \ldots$ requires only the values of the terms appearing before it. Thus, one can compute these values incrementally starting with $\mathsf{v}_{1,0}$. Furthermore, inserting a new vertex (say the $(n+1)$th vertex, v_n) to a simplex requires us to compute only the new coordinates $\mathsf{v}_{n,0}, \mathsf{v}_{n,1}, \ldots, \mathsf{v}_{n,n-1}$, for the Euclidean realization of the new extended simplex.

With the understanding that $\sum_{p=\alpha}^{\beta} h(p) = 0$ whenever $\beta < \alpha$, equations (7)–(10) can be written more compactly to give the coordinates of the embedded vertices as

$$\mathsf{v}_{j,k} = \begin{cases} \big(d_{j,0}^2 - d_{j,k+1}^2 + \mathsf{v}_{k+1,k}^2 + \\ \quad \sum_{p=0}^{k-1} (\mathsf{v}_{k+1,p}^2 - 2\mathsf{v}_{j,p}\mathsf{v}_{k+1,p}) \big) \,/\, 2\mathsf{v}_{k+1,k}, & k < j-1 \\ \pm \sqrt{d_{j,0}^2 - \sum_{p=0}^{j-2} \mathsf{v}_{j,p}^2}, & k = j-1 \\ 0, & k \ge j \end{cases} \tag{11}$$

We choose the positive solution for coordinates $\mathsf{v}_{j,j-1}$. The computation of $\mathsf{v}_{1,0},\ \mathsf{v}_{2,0}, \mathsf{v}_{2,1},\ \mathsf{v}_{3,0}, \mathsf{v}_{3,1}, \mathsf{v}_{3,2}, \ldots, \mathsf{v}_{j,0}, \mathsf{v}_{j,1}, \ldots, \mathsf{v}_{j,j-1},\ \mathsf{v}_{j+1,0}, \ldots, \ldots$ can be made in an incremental manner, with the computation of a term in this sequence requiring only the previous terms.

Lemma 4. *Equation* (11) *does not have a real solution if and only if the metric simplex,* (σ, d)*, is not Euclidean realizable.*

Lemma 5. *If* $\mathsf{v}_{j,j-1} = 0$ *for some j, and all the coordinates are real, then* (σ, d) *has a degenerate Euclidean realization.*

A.2. Coordinate computation for a point with given distances to $\{\mathbf{v}_j\}_{j=1,2,\ldots,n-1}$

Given a metric $(n-1)$-simplex, (σ, d), and the canonical Euclidean realization $\mathcal{E}_d(\sigma) = e : v_i \mapsto \mathbf{v}_i$, we consider an additional point, o, and its Euclidean realization in the same Euclidean space,

$$\mathbf{o} = [\mathsf{o}_0,\ \mathsf{o}_1, \ldots \cdots,\ \mathsf{o}_{n-2}] \in \mathbb{R}^{n-1} \tag{12}$$

Along with given its distances to all vertices in σ, except v_0: $\bar{d} : \sigma - v_0 \to \mathbb{R}_+$. For convenience, we write $\bar{d}(v_i) = \bar{d}_i$, $i = 1, 2, \ldots, n-1$. Thus, for $0 < l \le n-1$

$$\bar{d}_l^2 = \sum_{p=0}^{l-1} (\mathsf{o}_p - \mathsf{v}_{l,p})^2 + \sum_{p=l}^{n-2} \mathsf{o}_p^2 \tag{13}$$

Thus, for $l = 2, 3, \ldots, n-1$,

$$\begin{aligned}
\bar{d}_l^2 - \bar{d}_1^2 &= \textstyle\sum_{p=0}^{l-1} (\mathsf{o}_p - \mathsf{v}_{l,p})^2 + \sum_{p=l}^{n-2} \mathsf{o}_p^2 \\
&\qquad - (\mathsf{o}_0 - \mathsf{v}_{1,0})^2 - \textstyle\sum_{p=1}^{n-2} \mathsf{o}_p^2 \\
&= -2\mathsf{o}_0\mathsf{v}_{l,0} + \mathsf{v}_{l,0}^2 + 2\mathsf{o}_0\mathsf{v}_{1,0} - \mathsf{v}_{1,0}^2 \\
&\qquad + \textstyle\sum_{p=1}^{l-1} (-2\mathsf{o}_p\mathsf{v}_{l,p} + \mathsf{v}_{l,p}^2) \\
&= 2(\mathsf{v}_{1,0} - \mathsf{v}_{l,0})\,\mathsf{o}_0 - 2\textstyle\sum_{p=1}^{l-1} \mathsf{v}_{l,p}\mathsf{o}_p \\
&\qquad + \Big(\textstyle\sum_{p=0}^{l-1} \mathsf{v}_{l,p}^2 - \mathsf{v}_{1,0}^2 \Big) \\
\Rightarrow \quad \mathsf{o}_{l-1} &= \tfrac{\mathsf{v}_{1,0} - \mathsf{v}_{l,0}}{\mathsf{v}_{l,l-1}} \mathsf{o}_0 - \textstyle\sum_{p=1}^{l-2} \frac{\mathsf{v}_{l,p}}{\mathsf{v}_{l,l-1}} \mathsf{o}_p \\
&\qquad + \tfrac{1}{2\mathsf{v}_{l,l-1}} \Big(\textstyle\sum_{p=0}^{l-1} \mathsf{v}_{l,p}^2 - \mathsf{v}_{1,0}^2 + \bar{d}_1^2 - \bar{d}_l^2 \Big) \\
\Rightarrow \quad \mathsf{o}_k &= \alpha_{k,0}\mathsf{o}_0 + \textstyle\sum_{p=1}^{k-1} \alpha_{k,p}\mathsf{o}_p + \beta_k \\
&\qquad \text{(with } k = l-1\text{)}
\end{aligned} \tag{14}$$

where $\alpha_{k,0} = \frac{\mathsf{v}_{1,0}-\mathsf{v}_{k+1,0}}{\mathsf{v}_{k+1,k}}$ $\alpha_{k,p} = -\frac{\mathsf{v}_{k+1,p}}{\mathsf{v}_{k+1,k}}, p = 1,2,\ldots,k-1,$ and $\beta_k = \frac{1}{2\mathsf{v}_{k+1,k}}\left(\sum_{p=0}^{k}\mathsf{v}_{k+1,p}^2 - \mathsf{v}_{1,0}^2 + \overline{d}_1^2 - \overline{d}_{k+1}^2\right)$.

Letting

$$\mathsf{o}_l = A_l\mathsf{o}_0 + B_l \tag{15}$$

we have

$$\begin{aligned}\mathsf{o}_k &= \alpha_{k,0}\mathsf{o}_0 + \textstyle\sum_{p=1}^{k-1}\alpha_{k,p}\mathsf{o}_p + \beta_k \\ &= \alpha_{k,0}\mathsf{o}_0 + \textstyle\sum_{p=1}^{k-1}\alpha_{k,p}(A_p\mathsf{o}_0 + B_p) + \beta_k \\ &= \left(\alpha_{k,0} + \textstyle\sum_{p=1}^{k-1}\alpha_{k,p}A_p\right)\mathsf{o}_0 + \left(\beta_k + \textstyle\sum_{p=1}^{k-1}\alpha_{k,p}B_p\right)\end{aligned} \tag{16}$$

Thus, A_k and B_k can be determined iteratively as follows:

$$\begin{aligned} A_1 &= \alpha_{1,0}, & B_1 &= \beta_1 \\ A_k &= \alpha_{k,0} + \textstyle\sum_{p=1}^{k-1}\alpha_{k,p}A_p, & B_k &= \beta_k + \textstyle\sum_{p=1}^{k-1}\alpha_{k,p}B_p \end{aligned} \tag{17}$$

We can now determine o_0 from the expression for $\overline{d}_1^2$:

$$\begin{aligned}\overline{d}_1^2 &= (\mathsf{o}_0 - \mathsf{v}_{1,0})^2 + \textstyle\sum_{p=1}^{n-2}\mathsf{o}_p^2 \\ &= (\mathsf{o}_0^2 - 2\mathsf{v}_{1,0}\mathsf{o}_0 + \mathsf{v}_{1,0}^2) \\ &\qquad + \textstyle\sum_{p=1}^{n-2}(A_p^2\mathsf{o}_0^2 + 2A_pB_p\mathsf{o}_0 + B_p^2) \\ \Rightarrow \quad &\left(1 + \textstyle\sum_{p=1}^{n-2}A_p^2\right)\mathsf{o}_0^2 + 2\left(-\mathsf{v}_{1,0} + \textstyle\sum_{p=1}^{n-2}A_pB_p\right)\mathsf{o}_0 \\ &\qquad + \left(\mathsf{v}_{1,0}^2 - \overline{d}_1^2 + \textstyle\sum_{p=1}^{n-2}B_p^2\right) = 0\end{aligned} \tag{18}$$

We need to choose the solution of $\mathbf{o}$ such that $\mathbf{v}_0$ and $\mathbf{o}$ lies on the opposite sides of the hyperplane containing the points $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{n-1}$. Since we have chosen $\mathsf{v}_{1,0}$ to be positive, and since $\mathbf{v}_0$ is the origin, choosing the higher value of o_0 will satisfy this condition. Thus,

$$\mathsf{o}_0 = \frac{-V + \sqrt{V^2 - 4UW}}{2U} \tag{19}$$

where $U = \left(1 + \sum_{p=1}^{n-2}A_p^2\right)$, $V = 2\left(-\mathsf{v}_{1,0} + \sum_{p=1}^{n-2}A_pB_p\right)$, and $W = \left(\mathsf{v}_{1,0}^2 - \overline{d}_1^2 + \sum_{p=1}^{n-2}B_p^2\right)$.

Lemma 6. *Equation* (19) *does not have a real solution if and only if o is not embeddable in the same Euclidean space as an Euclidean realization of* (σ, d) *satisfying the distance relations* $\overline{d}$.

A.3. Intersection point of plane and line segment

A general point on the (affine) hyperplane, H_0, containing points $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{n-1}$ can be written as $\sum_{i=1}^{n-1} w_i\mathbf{v}_i$, where $\sum_{i=1}^{n-1} w_i = 1$. A point on the line, L, joining $\mathbf{v}_0$ and $\mathbf{o}$ can be written as $w_0\mathbf{v}_0 + w_o\mathbf{o}$ $(= w_o\mathbf{o})$ (since $\mathbf{v}_0 = [0,0,\ldots,0]$), where $w_0 + w_o = 1$. Thus, the point of intersection of H_0 and L is

$$\begin{aligned}&\mathbf{i}_0 = \textstyle\sum_{i=1}^{n-1} w_i\mathbf{v}_i = w_o\mathbf{o} \\ \Rightarrow &\textstyle\sum_{i=1}^{n-1} w_i\mathsf{v}_{i,j-1} = w_o\mathsf{o}_{j-1}, \quad \text{for } j = 1,2,\ldots,n-1 \\ \Rightarrow &\textstyle\sum_{i=j}^{n-1} w_i\mathsf{v}_{i,j-1} = w_o\mathsf{o}_{j-1} \quad (\text{since } \mathsf{v}_{i,j-1} = 0 \text{ for } i \le j-1) \\ \Rightarrow &w_j = \frac{w_o\mathsf{o}_{j-1} - \sum_{i=j+1}^{n-1} w_i\mathsf{v}_{i,j-1}}{\mathsf{v}_{j,j-1}} \\ \Rightarrow &w'_j = \frac{\mathsf{o}_{j-1} - \sum_{i=j+1}^{n-1} w'_i\mathsf{v}_{i,j-1}}{\mathsf{v}_{j,j-1}}, \quad \text{where } w'_j = \frac{w_j}{w_o}\end{aligned} \tag{20}$$

The above gives a recursive formula that lets us compute the terms in the sequence $w'_{n-1}, w'_{n-1}, \ldots, w'_1$ in an incremental manner, with computation of each term requiring the knowledge of the previous terms in the sequence only.

Since $\sum_{i=1}^{n-1} w_i = 1$ and $w'_j = \frac{w_j}{w_o}$, we have the following

$$w_j = \frac{w'_j}{\sum_{i=1}^{n-1} w'_i} \tag{21}$$

Sign of the Wights In general, let $H_{k_1,k_2,\ldots,k_p}$ be the $(n-1-p)$-dimensional hyperplane containing all the points in the set $\{\mathbf{v}_i\}_{i=0,1,\ldots,n-1} - \{\mathbf{v}_{k_j}\}_{j=1,2,\ldots,p}$ (i.e., the hyperplane of the subsimplex not containing $\mathbf{v}_{k_1}, \mathbf{v}_{k_2}, \ldots, \mathbf{v}_{k_p}$).

Lemma 7. *We have*

$$w_k \begin{cases} < 0 & \text{if and only if } \mathbf{i}_0 \text{ and } \mathbf{v}_k \text{ lie on the opposite sides of } H_{0,k} \text{ in } H_0 \\ = 0 & \text{if and only if } \mathbf{i}_0 \text{ lies on } H_{0,k} \\ > 0 & \text{if and only if } \mathbf{i}_0 \text{ and } \mathbf{v}_k \text{ lie on the same sides of } H_{0,k} \text{ in } H_0 \end{cases} \tag{22}$$

A.4. Computation of plane with given distances from $\{\mathbf{v}_j\}_{j=1,2,\ldots,n-1}$

Given a metric $(n-1)$-simplex, (σ, d), and the canonical Euclidean realization $\mathcal{E}_d(\sigma) = e : v_i \mapsto \mathbf{v}_i$, we consider a hyperplane, I, described by the equation, $\mathbf{u} \cdot \mathbf{x} + \mu = 0$, where $\mathbf{x} \in \mathbb{R}^{n-1}$ is a point on the hyperplane, $\mathbf{u} = [\mathsf{u}_0, \mathsf{u}_1, \ldots, \mathsf{u}_{n-2}] \in \mathbb{R}^{n-1}$ is an unit vector orthogonal to the plane, and μ is a constant.

Distance of the point $\mathbf{v}_j$ from the plane is $\overline{d}_j$. Thus, for $j = 1,2,\ldots,n-1$, we have

$$\begin{aligned}&\mathbf{u} \cdot \mathbf{v}_j + \mu = \overline{d}_j \\ \Rightarrow \quad &\sum_{p=0}^{j-1}\mathsf{u}_p\mathsf{v}_{j,p} + \mu = \overline{d}_j \\ \Rightarrow \quad &\mathsf{u}_{j-1} = \frac{\overline{d}_j - \mu - \sum_{p=0}^{j-2}\mathsf{u}_p\mathsf{v}_{j,p}}{\mathsf{v}_{j,j-1}} \\ \Rightarrow \quad &\mathsf{u}_k = \frac{\overline{d}_{k+1} - \mu - \sum_{p=0}^{k-1}\mathsf{u}_p\mathsf{v}_{k+1,p}}{\mathsf{v}_{k+1,k}}\end{aligned} \tag{23}$$

Letting

$$\mathsf{u}_l = M_l\,\mu + N_l \tag{24}$$

we have

$$\begin{aligned}\mathsf{u}_k &= \frac{\overline{d}_{k+1} - \mu - \sum_{p=0}^{k-1}(M_p\mu + N_p)\,\mathsf{v}_{k+1,p}}{\mathsf{v}_{k+1,k}} \\ &= -\left(1 + \frac{1}{\mathsf{v}_{k+1,k}}\sum_{p=0}^{k-1} M_p\mathsf{v}_{k+1,p}\right)\mu \\ &\quad + \left(\overline{d}_{k+1} - \frac{1}{\mathsf{v}_{k+1,k}}\sum_{p=0}^{k-1} N_p\mathsf{v}_{k+1,p}\right) \quad (25)\end{aligned}$$

Thus, we have the following recursive equation for M_k and N_k,

$$\begin{aligned}M_k &= -\left(1 + \frac{1}{\mathsf{v}_{k+1,k}}\sum_{p=0}^{k-1} M_p\mathsf{v}_{k+1,p}\right) \\ N_k &= \left(\overline{d}_{k+1} - \frac{1}{\mathsf{v}_{k+1,k}}\sum_{p=0}^{k-1} N_p\mathsf{v}_{k+1,p}\right) \quad (26)\end{aligned}$$

for $k = 0, 1, 2, \ldots, n-2$. With the understanding that $\sum_{p=\alpha}^{\beta} h(p) = 0$ whenever $\beta < \alpha$, we have $M_0 = -1$, $N_0 = \overline{d}_1$.

Since $\mathbf{u}$ is an unit vector, $\sum_{j=0}^{n-1} \mathsf{u}_j^2 = 1$. Thus, $\left(\sum_{j=0}^{n-1} M_j^2\right)\mu^2 + \left(2\sum_{j=0}^{n-1} M_jN_j\right)\mu + \left(\sum_{j=0}^{n-1} N_j^2 - 1\right) = 0$. Hence,

$$\mu = \frac{-Q + \sqrt{Q^2 - 4PR}}{2P} \quad (27)$$

where $P = \sum_{j=0}^{n-1} M_j^2$, $Q = 2\sum_{j=0}^{n-1} M_jN_j$, and $R = \sum_{j=0}^{n-1} N_j^2 - 1$. We choose the positive sign before the square root since we want the plane satisfying (23) that is farthest from the origin.

Equations (26), (27), and (24) compute the required plane, I, which is at a distance $\overline{d}_j$ from $\mathbf{v}_j, j = 1, 2, \ldots, n-1$.

Similar to what was described in Appendix A.3, the point at which the perpendicular (in the direction of $\mathbf{u}$) dropped from $\mathbf{v}_0$ onto the hyperplane I intersects the hyperplane H_0 is given by $\mathbf{i}_0 = \sum_{i=1}^{n-1} w_i\mathbf{v}_i$, with

$$w_k = \frac{w'_k}{\sum_{i=1}^{n-1} w'_i}, \quad k = 1, 2, \ldots, n-1 \quad (28)$$

where w'_j can be computed recursively using the formula $w'_j = \frac{\mathsf{u}_{j-1} - \sum_{i=j+1}^{n-1} w'_i\mathsf{v}_{i,j-1}}{\mathsf{v}_{j,j-1}}$.

A.5. Sketch of proof: $\overline{d}$-value approaches path metric on Riemannian manifold.

The key to proving Proposition 7 is to show that for a simplex with edge lengths $O(\epsilon)$, the error between the extrapolated $\overline{d}$-distance and the *true*/exact distance function on the manifold, M, is $O(\epsilon^2)$. For the proof we use an inductive argument: assuming that this error is $O(\epsilon^2)$ for the base vertices $v_1, v_2, \ldots, v_{n-1}$ of a simplex, we prove that the extrapolation, $\overline{d}(v_0)$, has at most an error of $O(\epsilon^2)$.

Suppose $\sigma = \{v_0, v_1, \ldots, v_{n-1}\}$, $d : \sigma \times \sigma \to \mathbb{R}_{\geq 0}$, is a metric simplex on the configuration manifold, M. The canonical Euclidean realization of the simplex, $e : \sigma \to \mathbb{R}^{n-1}$, constructs a local coordinate chart on the manifold such that the matrix representation of the Riemannian metric tensor on this chart is the identity matrix somewhere in the neighborhood of the simplex since the metric on the canonical Euclidean space, $\mathbb{R}^{n-1}$, is the Euclidean metric (Jost, 2008).

Now consider $\widetilde{d} : M \to \mathbb{R}_{\geq 0}$ to be the *true*/exact distance function on the configuration manifold (from some start point in M). Using the local coordinate chart, e, one can write a Taylor expansion of $\widetilde{d}$, centered at, say, v_1:

$$\widetilde{d}(\mathbf{v}) = \widetilde{d}(\mathbf{v}_1) + \sum_{i=1}^{n-1}\frac{\partial\widetilde{d}}{\partial x_i}\Delta x_i + O(\epsilon^2)$$

where $x_i, i = 1, 2, \ldots, n-1$ are the coordinates in the coordinate chart, Δx_i is the difference of the ith coordinate between the points $\mathbf{v}$ and $\mathbf{v}_1$, and the partial derivatives are computed at $\mathbf{v}_1$. Note that the quadratic terms in the Taylor series have terms of the form $\Delta x_i\Delta x_j$ which is $O(\epsilon^2)$ since the lengths of the edges of the simplex are of $O(\epsilon)$. Also note that $\nabla_x\widetilde{d} := [\frac{\partial\widetilde{d}}{\partial x_1}, \frac{\partial\widetilde{d}}{\partial x_2}, \ldots, \frac{\partial\widetilde{d}}{\partial x_{n-1}}]$ is the coordinate representation of the gradient of the distance function.

What either the *spherical* or the *linear* extrapolation methods do is numerically estimate the coordinate representation of the gradient vector, $\nabla_x\widetilde{d}$ (the $\hat{\mathbf{o}}$ for the spherical, and $\mathbf{u}$ for the linear).[2] However, they do that by numerically computing $\nabla_x\overline{d}$, since $\overline{d}$ is the value that has been computed for $v_1, v_2, \ldots, v_{n-1}$. The linear method is, in particular, easy to understand since it precisely solves for $\mathbf{u}$ in an equation of the form $\overline{d} = \alpha + \mathbf{u}\cdot\Delta\mathbf{x}$ (see Appendix A.4), where $\mathbf{u}$ represents the gradient. Since this is a first-order approximation (Burden and Faires, 1989), the error accumulated in this gradient computation is $O(\epsilon)$. This, along with the base case for our inductive argument, $\overline{d}(v_i) = \widetilde{d}(v_i) + O(\epsilon^2)$, $i = 1, 2, \ldots, n-1$ (note that in the gradient estimation we only use the previously estimated $\overline{d}$-distances at $v_1, v_2, \ldots, v_{n-1}$), gives us the estimated gradient of the distance function, $\nabla_x\overline{d}^{\text{estimated}} = \nabla_x\widetilde{d} + O(\epsilon)\,\hat{\mathbf{f}}$ (for some unit vector $\hat{\mathbf{f}}$). Thus,

$$\begin{aligned}\widetilde{d}(\mathbf{v}) - \overline{d}(\mathbf{v}) &= \left(\widetilde{d}(\mathbf{v}_1) - \overline{d}(\mathbf{v}_1)\right) \\ &\quad + \left(\nabla_x\widetilde{d} - \nabla_x\overline{d}^{\text{estimated}}\right)\cdot\Delta\mathbf{x} + O(\epsilon^2) \\ &= O(\epsilon^2) + O(\epsilon)\,\hat{\mathbf{f}}\cdot\Delta\mathbf{x} + O(\epsilon^2) \\ &\quad \text{[The first substitution comes from the base case of the inductive assumption, while the second substitution from the above discussion.]} \\ &= O(\epsilon^2) \quad [\text{Since } \epsilon = \max_i \Delta x_i.]\end{aligned}$$

Specializing this result to $\mathbf{v} = \mathbf{v}_0$ gives the required condition that the error in the extrapolated $\overline{d}$-distance is of $O(\epsilon^2)$.

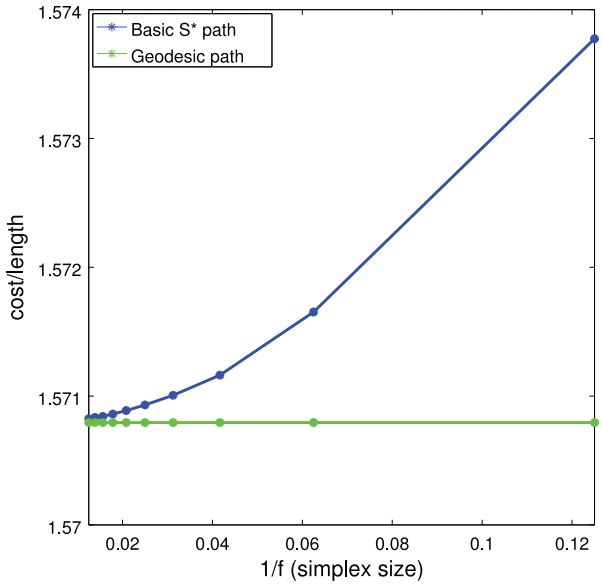


Fig. 18. The error in the cost computed by Basic S* decays quadratically as the discretization size is reduced. This is the same data shown in Figure 13d.

To further validate the claim in this proposition, we closely investigate the error in the cost of paths computed on the surface of a sphere (illustrated in Section 5 and in Figure 13) as a function of the size of the simplices. Figure 18 compares the cost of paths computed using Basic S* with the analytically computed length of the geodesic path as a function of the simplex size (which is proportional to the reciprocal of the fineness, f). As can be observed, the quadratic relationship of the error with the simplex size is evident.

It is also worth noting that the above proof relies on differentiability of the distance function. As long as the regions where the distance function is not differentiable constitute a set of measure zero in the configuration manifold, the result still holds true. This is true for Riemannian manifolds with punctures/obstacles that have a finite number of corner points (the distance function is not differentiable exactly at the corner points). However, this suboptimality due to corner points can be reduced by refining the discretization around corners (so that the regions/simplices where the distance function is non-smooth is reduced). This is illustrated in the result of Figure 12(k). Furthermore, if there is no non-uniformity in the metric/cost-function in the underlying configuration manifold and hence in the global simplicial complex, and if there are no corner points (due to obstacles), then the errors will be precisely zero (i.e., the local Euclidean embedding of each simplex matches exactly with the metric of the underlying configuration manifold): that is the trivial case. The $O(\epsilon^2)$ error is non-trivial (and, thus, the statement of the proposition is meaningful) only when the cost function/metric is non-uniform.