

Multi-Query Shortest-Path Problem in Graphs of Convex Sets

Savva Morozov, Tobia Marcucci, Alexandre Amice, Bernhard Paus Graesdal,
Rohan Bosworth, Pablo A. Parrilo, and Russ Tedrake

Massachusetts Institute of Technology

Abstract. The Shortest-Path Problem in Graph of Convex Sets (SPP in GCS) is a recently developed optimization framework that blends discrete and continuous decision making. Many relevant problems in robotics, such as collision-free motion planning, can be cast and solved as an SPP in GCS, yielding lower-cost solutions and faster runtimes than state-of-the-art algorithms. In this paper, we are motivated by motion planning of robot arms that must operate swiftly in static environments. We consider a multi-query extension of the SPP in GCS, where the goal is to efficiently precompute optimal paths between given sets of initial and target conditions. Our solution consists of two stages. Offline, we use semidefinite programming to compute a coarse lower bound on the problem’s cost-to-go function. Then, online, this lower bound is used to incrementally generate feasible paths by solving short-horizon convex programs. For a robot arm with seven joints, our method designs higher quality trajectories up to two orders of magnitude faster than existing motion planners.

Keywords: Control Theory and Optimization · Motion and Path Planning · Collision Avoidance

1 Introduction

A Graph of Convex Sets (GCS) [24] is a graph where each vertex is paired with a convex set and an optimization variable inside this set, while each edge couples adjacent vertex variables through additional convex costs and constraints. In the Shortest-Path Problem (SPP) in GCS [26], we simultaneously seek a discrete path through this graph and optimize the continuous variables associated with the vertices along the path, while minimizing the cumulative edge costs.

Though the SPP in GCS is NP-hard [26, Section 9.2], effective solution methods have been proposed in [26,8]. This technique has shown remarkable success in various robotics applications, such as optimal control [26], planning through contact [13], and other robotics problems [30,19,9]. In real-world hardware deployment, it has been especially effective in collision-free motion planning [25], addressing the challenges of non-convex obstacle avoidance constraints. However,

This work was supported by Amazon.com Services LLC, PO No. 2D-12585006; The AI Institute; Dexai Robotics; National Science Foundation, DMS-2022448, UC Berkeley, 00010918. Corresponding author is Savva Morozov, savva@mit.edu.

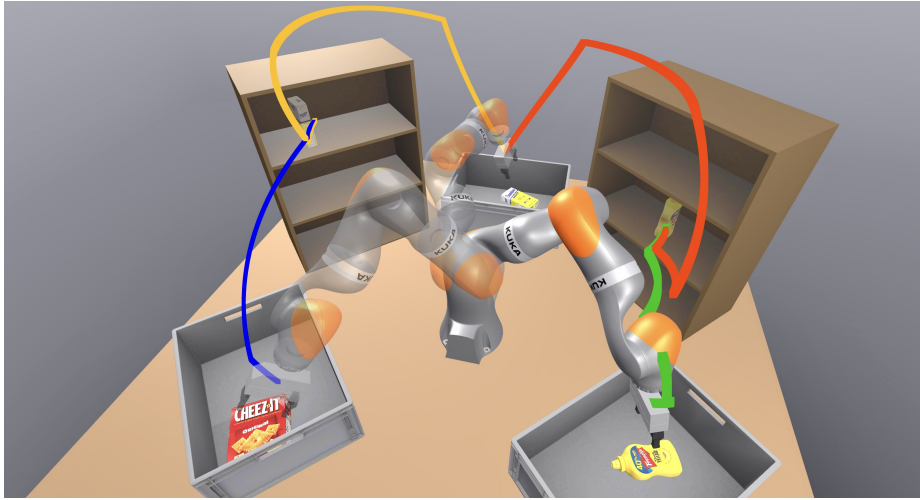
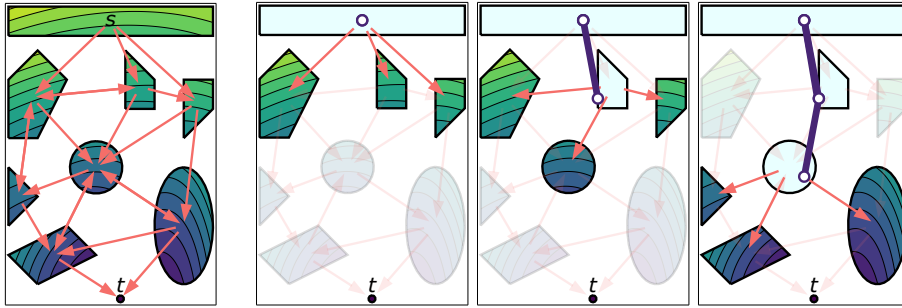


Fig. 1: Robotic arm in a simulated environment, tasked with moving items between shelves and bins. Shown are four queries for collision-free motion planning.

solving the SPP in GCS can sometimes be too slow for real-time applications on high-dimensional robotic systems. Consider a 7-DoF KUKA iiwa robot arm repeatedly performing online motion planning in a static environment. When the environment is simple, the GCS is small, and the shortest path queries can be solved quickly, in under 50ms [25]. However, when the environment is complex and the configuration space must be covered thoroughly, as in Fig. 1, the GCS becomes large, and the shortest path queries can take up to of 600ms. This is not practical for high-productivity applications, such as robot arms in warehouses, where the company’s income is nearly proportional to the operational speed.

In an effort to reduce solve times for online shortest path queries in GCS, we seek an efficient way of precomputing optimal paths between given sets of source and target conditions in the GCS. We formulate this problem as a generalization of the SPP in GCS that is akin to the all-pairs generalization of the classical SPP. Our solution contains two phases, illustrated in Fig. 2. Offline, we solve a semidefinite program that produces convex quadratic lower bounds to the cost-to-go function over the convex sets associated with GCS vertices. Pictured in Fig. 2a are the contour plots of these lower bounds at every vertex. Then, online, we use a greedy multi-step lookahead policy with the cost-to-go lower bounds to determine the next vertex to visit. Thus, as shown in Fig. 2b, the path is obtained incrementally, one vertex at a time. Though the quadratic cost-to-go lower bounds can be coarse, using the lookahead policy is equivalent to producing piecewise-quadratic lower bounds, which can be very expressive. As a result, the obtained paths are nearly optimal in practice. Convexity of the quadratic cost-to-go lower bounds allows us to evaluate the greedy policy by solving a set of small convex programs in parallel, which can be done quickly at runtime. Applied



(a) Offline: synthesize cost-to-go over the GCS. Contour plots are shown.

(b) Online: at each iteration, we evaluate all n -step paths from the current vertex ($n=1$ shown) and greedily select the decision that minimizes the n -step lookahead cost-to-go. The first three iterations are shown, as the path is built incrementally.

Fig. 2: Illustration of our approach. The GCS instance is embedded in \mathbb{R}^2 , with the source vertex at the top and the target vertex at the bottom. The edges are shown as red arrows, and the edge length is the squared Euclidean distance.

to the complex scenario shown in Fig. 1, our method requires just 6s of offline computation to produce the cost-to-go lower bounds. Subsequent online queries take 2-11ms, which is up to two orders of magnitude faster than solving the SPP in GCS from scratch.

1.1 Literature review

Graph search plays a central role in both modelling and solving a wide variety of planning problems in robotics. In this section we briefly connect our work to some notable examples in this literature.

A common approach to motion planning is to construct a graph where nodes correspond to collision-free configurations and edges correspond to collision-free motions. The most popular approaches based on this idea are the Rapidly exploring Random Trees (RRT) [22], Probabilistic Roadmap (PRM) [17], and their many variants [18,6,14,16]. The GCS approach to motion planning is similar to the PRM one, but collision-free configurations are replaced with large collision-free sets [25]. GCS avoids two major drawbacks of planning with a PRM: the need to densely sample in high-dimensional spaces and post-process the motion plan to obtain a smooth trajectory. However, generating these collision-free sets can be computationally challenging and expensive. Furthermore, SPP in GCS queries can still be very expensive, motivating this current work.

The importance of the SPP has led to a breadth of literature on its solution, with Bellman’s dynamic programming approach illustrating the central role of the cost-to-go function [2,4]. Given the cost-to-go, a shortest path can be extracted using a simple greedy strategy: given a vertex v , the next vertex in the path is the one which minimizes the cost-to-go among all the neighbors of v . This is captured by the famous Bellman’s equation[2].

Multi-query SPP setting has also been thoroughly investigated. The All-Pairs Shortest Paths (APSP) is the problem of finding shortest paths between every pair of vertices in a discrete graph [10, Ch. 23]. One method for solving this problem, from which we draw particular inspiration, first computes the cost-to-go function between every pair of vertices in the graph (also known as a *distance oracle*). This cost-to-go is used to produce a successor along the shortest path between every pair of vertices, which is stored into the *successor matrix*. The optimal paths are retrieved by sequentially querying this matrix.

Explicit solutions to the Bellman equation exist only in a handful of contexts. In the case of purely discrete graphs, a number of efficient methods exist [12,35,15], where the cost-to-go function can be encoded using a simple matrix. Another notable example from control is Explicit Model Predictive Control (MPC) where the cost-to-go is a piecewise quadratic [3]. However, even in these settings, storing the cost-to-go can be prohibitively expensive for large graphs, particularly in the APSP setting. In the purely discrete setting, the description of the APSP cost-to-go function grows quadratically in the size of the graph, while in the MPC setting it grows exponentially.

In most cases, solving the Bellman equation is known to be intractable. This has motivated a breadth of literature for computing approximations for the cost-to-go in various setting [11,32,21,4,23,34]. Similarly, in this paper we seek a computationally tractable way to approximate the cost-to-go function to solve the APSP in GCS. The generalization in the particular context of GCS is not straightforward and constitutes one of the contributions of this work.

2 All-Pairs Shortest Paths in a Graph of Convex Sets

We seek to efficiently precompute optimal solutions to the SPP in GCS between given sets of source and target conditions. Section 2.1 presents the classical APSP, which is the corresponding problem in an ordinary graph. In Section 2.2, we describe the single-query SPP in GCS. We then formulate the APSP in GCS in Section 2.3, and outline our approximate solution method in Section 2.4.

2.1 All-Pairs Shortest Paths

Graphs and paths. Let $G = (\mathcal{V}, \mathcal{E})$ be a directed graph with vertex set \mathcal{V} and edge set \mathcal{E} . Given a source vertex s and target vertex t , an s - t path is a sequence of distinct vertices $p = (s = v_0, v_1, \dots, v_K = t)$, where each consecutive pair of vertices is connected by an edge in \mathcal{E} and no vertex is revisited. We define $\mathcal{E}_p = \{(v_0, v_1), \dots, (v_{K-1}, v_K)\}$ as the set of edges traversed by the path p , and denote the set of all s - t paths in G as $\mathcal{P}_{s,t}$.

Shortest Path Problem (SPP). Let us associate with every edge $e \in \mathcal{E}$ a non-negative edge cost $c_e \in \mathbb{R}_+$. A shortest path p between the vertices s and t minimizes the sum of the edge costs along the path:

$$\min_p \sum_{e \in \mathcal{E}_p} c_e \quad \text{s.t.} \quad p \in \mathcal{P}_{s,t}.$$

The optimal value of this program is called the *cost-to-go* between s and t , and is denoted by $J_{s,t}^*$. *The principle of optimality* [2] holds in this context, stating that every subpath of a shortest path is itself a shortest path. This forms the foundation for many efficient solution algorithms to this problem.

All-Pairs Shortest Paths (APSP). The APSP is the multi-query generalization of the SPP, where we seek a shortest path between all pairs of vertices in a graph. Efficient solutions to the APSP leverage the principle of optimality. Instead of computing the full path for each pair of vertices, it suffices to compute only the immediate successor along this path. The full path can thus be attained incrementally, one vertex at the time.

This solution to the APSP can be implicitly encoded via the cost-to-go function $J_{v,t}^*$ for every pair of vertices v and t , computed via dynamic programming [10, Ch. 23] [12,35,15]. The successor is then computed by greedily picking a vertex that minimizes the one-step lookahead with respect to the cost-to-go:

$$\pi(v, t) = \arg \min_w c_e + J_{w,t}^* \quad (1a)$$

$$\text{s.t. } e = (v, w) \in \mathcal{E}. \quad (1b)$$

The solution π is a decision policy that, given the current and target vertices v and t , selects the next vertex on the shortest v - t path. We refer to π as the *successor policy*.

2.2 Shortest-Path Problem in a Graph of Convex Sets

Graph of Convex Sets. A GCS is a directed graph $G = (\mathcal{V}, \mathcal{E})$, where each vertex $v \in \mathcal{V}$ is paired with a bounded convex set \mathcal{X}_v and a continuous variable $x_v \in \mathcal{X}_v$. Each edge $e = (v, w) \in \mathcal{E}$ is then paired with a convex set $\mathcal{X}_e \subseteq \mathcal{X}_v \times \mathcal{X}_w$ and a convex non-negative edge length function $l_e : \mathcal{X}_e \rightarrow \mathbb{R}_+$, such that the adjacent vertex variables satisfy the constraint $(x_v, x_w) \in \mathcal{X}_e$, while minimizing the length $l_e(x_v, x_w)$ [24].

The Shortest Path Problem in a Graph of Convex Sets. The SPP in GCS between point $\bar{x}_s \in \mathcal{X}_s$ of vertex s and $\bar{x}_t \in \mathcal{X}_t$ of vertex t is defined as follows:

$$\min_{p, \{x_v\}_{v \in p}} \sum_{e=(v,w) \in \mathcal{E}_p} l_e(x_v, x_w) \quad (2a)$$

$$\text{s.t. } p \in \mathcal{P}_{s,t}, \quad (2b)$$

$$x_s = \bar{x}_s, \quad x_t = \bar{x}_t, \quad (2c)$$

$$x_v \in \mathcal{X}_v, \quad \forall v \in p, \quad (2d)$$

$$(x_v, x_w) \in \mathcal{X}_e, \quad \forall e = (v, w) \in \mathcal{E}_p. \quad (2e)$$

Similar to the classical SPP, the SPP in GCS searches for an s - t path $p = (v_0, v_1, \dots, v_K)$ though a graph, which is a sequence of distinct vertices. In addition to that, it also searches for a sequence of corresponding vertex variables

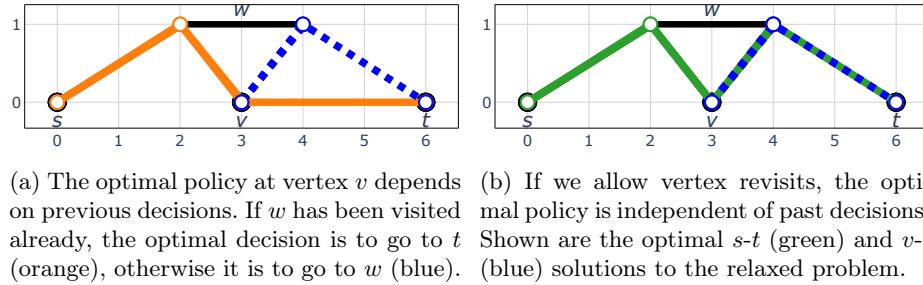


Fig. 3: The two-dimensional GCS from Example 1. The convex sets paired with s, v, t are points and the one paired with w is a segment. The GCS is fully connected, and the edge lengths are the squared Euclidean distance.

($\bar{x}_s = x_{v_0}, x_{v_1}, \dots, x_{v_K} = \bar{x}_t$), referred to as a *trajectory*. This trajectory satisfies the vertex and edge constraints (2c), (2d), (2e), while minimizing the edge costs (2a). The optimal solution to (2) is thus a tuple (path and trajectory). We denote the optimal value of (2) as $J_{s,t}^*(\bar{x}_s, \bar{x}_t)$ and refer to it as the *cost-to-go* from point \bar{x}_s of vertex s to point \bar{x}_t of vertex t .

Unlike the classical SPP, the SPP in GCS is NP-hard [24, Section 9.2], and thus unlikely to have a polynomial-time solution. However, it can be reformulated as a Mixed-Integer Convex Program (MICP) with a strong convex relaxation [26]: using a rounding strategy from [25], this relaxation often yields near-optimal solutions in practice.

For the classical SPP, the principle of optimality holds and the optimal policy is independent of past decisions, which simplifies the problem and enables many efficient solution algorithms. As demonstrated in the following example, these properties break down in the SPP in GCS.

Example 1. Consider the GCS in Fig. 3, which is embedded in \mathbb{R}^2 . This GCS has four vertices $\mathcal{V} = \{s, v, w, t\}$, where the convex sets $\mathcal{X}_s, \mathcal{X}_v, \mathcal{X}_t$ are points, and the convex set \mathcal{X}_w is a segment. Every vertex is connected to every other vertex with an edge, and the edge lengths l_e are the squared Euclidean distance (e.g., $l_{(v,w)} = \|x_v - x_w\|_2^2$).

Due to the constraint that vertices cannot be revisited, the optimal policy is a function of the set of previously visited vertices. This is demonstrated in Fig. 3a, where we plot the optimal $s-t$ path in orange and the optimal $v-t$ path in blue. The optimal decision at vertex v depends on previously visited vertices: if w was visited before, the optimal decision is to go to t (orange), otherwise the optimal decision is to go to w (blue).

Observe also that the principle of optimality does not hold for this problem: the $v-t$ subpath of the optimal $s-t$ path (orange) is not the optimal $v-t$ path (blue). We cannot substitute the optimal $v-t$ path (blue) in place of the original $v-t$ subpath, since the resulting vertex sequence (s, w, v, w, t) (Fig. 3b, green) visits vertex w twice, and is therefore not a path.

The constraint that vertices cannot be revisited is a key challenge of the SPP in GCS. This is unlike the classical SPP with non-negative edge lengths, where this constraint does not increase problem complexity. It can be shown that if we allow vertex revisits, then the principle of optimality holds, and the optimal decision policy is independent of past decisions. This is illustrated in Fig. 3b, where the optimal s - t and v - t solutions to the relaxed problem are shown in green and blue respectively.

2.3 All-Pairs Shortest Paths in a Graph of Convex Sets

The APSP in GCS extends the classical APSP in a natural way. We are given a set of source vertices $\mathcal{S} \subset \mathcal{V}$ and a set of target vertices $\mathcal{T} \subset \mathcal{V}$. The goal is to solve the SPP in GCS between every pair of source and target points $\bar{x}_s \in \mathcal{X}_s$ and $\bar{x}_t \in \mathcal{X}_t$, and every pair of source and target vertices $s \in \mathcal{S}$ and $t \in \mathcal{T}$. Since the SPP in GCS is NP-hard, the APSP in GCS is at least NP-hard as well.

2.4 Method outline

Our approach generalizes the solution to the classical APSP outlined in Section 2.1. We proceed in two phases. Offline, we compute a coarse quadratic lower bound on the cost-to-go between relevant pairs of GCS vertices. Then online, we extend the greedy policy (1) to the GCS setting. At runtime, we rollout this policy to obtain the solution path incrementally, one vertex at a time.

Unlike the classical APSP, a greedy policy with the cost-to-go $J_{s,t}^*$ is not an optimal policy for the APSP in GCS. This is because the optimal policy for paths in GCS depends on previously visited vertices, which is not captured by the cost-to-go $J_{s,t}^*(x_s, x_t)$. Thus, our approach is bound to yield approximate solutions, further limited by the coarseness of quadratic cost-to-go lower bounds.

To incorporate the challenging “no-vertex-revisit constraint” into the cost-to-go function, we relax this constraint by introducing penalties for vertex revisits. These penalties are applied to the edge lengths, producing a biased cost-to-go lower bound that discourages revisits. When rolling out a greedy policy online, we also explicitly prohibit vertex revisits. To mitigate the coarseness of quadratic cost-to-go lower bounds and better approximate the optimal policy, we employ a multi-step lookahead generalization of the greedy policy (1), optimizing over n -step decision sequences at each iteration.

3 Offline phase: synthesis of cost-to-go lower bounds

In Section 3.1, we present the optimization problem that produces cost-to-go lower bounds for the APSP in GCS. This program is infinite-dimensional, so in Section 3.2 we present a tractable numerical approximation for it.

For clarity of presentation, we make some simplifying assumptions. First, we assume that we have just one source vertex and one target vertex, i.e., $\mathcal{S} = \{s\}$ and $\mathcal{T} = \{t\}$. Second, we assume that the set \mathcal{X}_t corresponding to the target

vertex t is a singleton: $\mathcal{X}_t = \{x_t\}$. Since the target vertex t and point x_t are fixed, we also simplify the notation and refer to $J_{v,t}^*(x_v, x_t)$ as $J_v^*(x_v)$. The extensions of our method when these assumptions do not hold are straightforward and discussed in Appendix A.

3.1 Cost-to-go lower bounds via infinite-dimensional LP

The cost-to-go lower bounds are synthesized with the following optimization problem:

$$\begin{aligned} \max_{\{J_v, h_w\}_{v \in \mathcal{V}}} & \int_{\mathcal{X}_s} J_s(x) d\phi_s(x) & (3a) \\ \text{s.t.} & J_v : \mathcal{X}_v \rightarrow \mathbb{R}, & \forall v \in \mathcal{V}, & (3b) \\ & h_w \geq 0 & \forall w \in \mathcal{V}, & (3c) \\ & J_v(x_v) \leq l_e(x_v, x_w) + h_w + J_w(x_w), & \forall e = (v, w) \in \mathcal{E}, & (3d) \\ & & \forall (x_v, x_w) \in \mathcal{X}_e, & \\ & J_t(x_t) = - \sum_{w \in \mathcal{V}} h_w. & (3e) \end{aligned}$$

We now give a detailed line-by-line explanation of this program, and prove the validity of the lower bounds it produces in Lemma 1 below.

In constraint (3b), we associate with every vertex $v \in \mathcal{V}$ a (possibly non-convex) function J_v defined over the set \mathcal{X}_v . These functions serve as the lower bounds on the cost-to-go J_v^* , as will be shown later. We emphasize that we are searching over the space of functions J_v , not over the individual points x_v .

In the objective function (3a), ϕ_s is a probability distribution of anticipated source conditions over the set \mathcal{X}_s . Thus, the integral in (3a) maximizes the weighted average of J_s over the source set \mathcal{X}_s , effectively “pushing up” on the cost-to-go lower bound at the source vertex.

In (3c), we introduce a non-negative penalty h_w for every vertex $w \in \mathcal{V}$. This penalty is meant to discourage revisits to vertex w , which is a way to relax the constraint that a path must not visit any vertex more than once.

To implement the penalty h_w , we increment the edge length l_e for every edge $e \in \mathcal{E}$ that enters vertex w . This is formalized in (3d), which states that for every edge $e = (v, w)$ and a feasible transition $(x_v, x_w) \in \mathcal{X}_e$, the value $J_v(x_v)$ is a lower bound on the sum of the penalty-incremented edge length $l_e(x_v, x_w) + h_w$ and the subsequent cost-to-go lower bound $J_w(x_w)$. As written, the non-negative penalty h_w increases the cost of the edges leading into vertex w , thereby discouraging visits to w . However, since our goal is to only discourage vertex revisits, we need to waive the penalty h_w once. This is achieved by setting the cost-to-go lower bound $J_t(x_t)$ to $-\sum_{w \in \mathcal{V}} h_w$ in constraint (3e). Upon reaching the target vertex, we subtract the sum of all vertex penalties from the cost-to-go lower bound, effectively waiving the penalties once per vertex. We now show that these constraints produce lower bounds on the cost-to-go function.

Lemma 1. *Let J_v and h_v for $v \in \mathcal{V}$ be a feasible solution of problem (3). Then*

$$J_v(x_v) \leq J_v^*(x_v) \text{ for all } v \in \mathcal{V}.$$

Proof. Consider the optimal solution to program (3), and let v be some vertex. Let p be an optimal path from a point $x_v \in \mathcal{X}_v$ to the target point x_t . Since p is a path, it contains no repeated vertices. Adding the constraint (3d) along the edges \mathcal{E}_p of this optimal path, we have:

$$J_v(x_v) \leq \sum_{e=(u,w) \in \mathcal{E}_p} l_e(x_u, x_w) \sum_{w \in p} h_w + J_t(x_t), \quad (4)$$

where x_u, x_w are the vertex variables of the optimal trajectory corresponding to p . Constraint (3e) states that $J_t(x_t) = -\sum_{w \in \mathcal{V}} h_w$, while the sum of the edge lengths $l_e(x_u, x_w)$ along the optimal path p is by definition the cost-to-go $J_v^*(x_v)$. Substituting and rearranging terms, we obtain:

$$J_v(x_v) + \sum_{w \notin p} h_w \leq J_v^*(x_v). \quad (5)$$

Since the penalties h_w are non-negative by (3c), the conclusion follows. \square

By maximizing the weighted average of J_s in the objective function (3a), the program (3) seeks the best possible lower bound J_s on the cost-to-go J_s^* , up to the relaxation gap introduced by the vertex penalties. This gap is clear from (5): for $x_s \in \mathcal{X}_s$, the sum of the off-the-optimal-path penalty terms $\sum_{w \notin p} h_w$ need not to be zero, so $J_s(x_s)$ need not be a tight lower bound on $J_s^*(x_s)$. In other words, recall that, upon reaching the target, we waive the penalties h_w for every vertex $w \in \mathcal{V}$. As a result, we do not just waive the first-time penalties on vertices along the optimal path p , we also waive the off-the-path penalties $\sum_{w \notin p} h_w$, which were never accrued in the first place. Waiving these off-the-path penalties introduces the gap between J_s and J_s^* .

Example 1, continued. Consider the solution to program (3) for the GCS instance in Fig. 3. Setting the revisit penalty $h_w = 0$ results in $J_s = 14$, which is the cost of the vertex sequence that visits w twice (green in Fig. 3b). By jointly optimizing over the penalties and the cost-to-go lower bounds, program (3) selects the penalty $h_w = 2$. Revisiting vertex w is no longer advantageous, and the cost of the shortest s - t path (orange in Fig. 3a) is achieved: $J_s = J_s^* = 16$.

We note that program (3) naturally generalizes the cost-to-go synthesis LP for the classical SPP [11]. When each convex set \mathcal{X}_v is a singleton, the problem reduces to the classical SPP, where functions J_v are defined at single points and represented by a single decision variable. Setting vertex penalties $h_w = 0$ recovers the standard cost-to-go synthesis LP for the classical SPP:

$$\begin{aligned} \max_{\{J_v\}_{v \in \mathcal{V}}} \quad & J_s \\ \text{s.t.} \quad & J_v \leq l_e + J_w, \quad \forall e = (v, w) \in \mathcal{E}, \\ & J_t = 0. \end{aligned} \quad (6)$$

Compared to the purely discrete setting of (6), optimization program (3) is also an LP; however, it searches over the space of functions and is therefore infinite-dimensional. Next, we develop a tractable finite-dimensional approximation to (3) that is conducive to numerical methods.

3.2 Numerical approximation via semidefinite programming

We now produce an approximate solution to the cost-to-go synthesis program (3). We restrict each function J_v to be convex quadratic, which allows us to cast (3) as a tractable Semidefinite Program (SDP). SDPs are mathematical programs where the objective function is linear and the constraints are either linear or linear matrix inequalities (LMIs). To help with the presentation, we first state without proof three well-known facts.

Lemma 2 (e.g., [5, App. A.1]). *A quadratic function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is non-negative if and only if it is representable as a Positive-Semidefinite (PSD) quadratic form:*

$$f(x) = \begin{bmatrix} 1 \\ x \end{bmatrix}^\top Q \begin{bmatrix} 1 \\ x \end{bmatrix} \text{ for some } Q \succeq 0.$$

Lemma 3 ([5, Section 3.2.4]). *Let $\mathcal{X} = \{x \in \mathbb{R}^n \mid g_i(x) \geq 0, \forall i = 1, \dots, m\}$. The function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is non-negative on the set \mathcal{X} if there exists $\lambda \in \mathbb{R}_+^m$, such that $f(x) - \sum_{i=1}^m \lambda_i g_i(x)$ is non-negative for every $x \in \mathbb{R}^n$.*

Corollary 1. *Suppose that in Lemma 3, the function f is quadratic, and all g_i functions are affine or convex quadratic. Then we can apply Lemma 2 to verify Lemma 3 via an LMI, i.e., we can verify if f is non-negative over \mathcal{X} by searching for a PSD matrix in an affine subspace.*

Using these facts, we proceed to cast program (3) as an SDP.

Defining cost-to-go lower bounds in (3b). We restrict lower bounds J_v per vertex $v \in \mathcal{V}$ to be convex quadratic functions. By Lemma 2, searching for such functions is equivalent to searching for appropriate PSD matrices Q_v . The decision variables are thus the coefficients of the quadratic polynomials. As a result, we produce coarse quadratic lower bounds on the optimal J_v^* ; this coarseness will be mitigated via the multi-step lookahead policies.

Constraint (3c) is already linear, and constraint (3e) is linear in the coefficients of the quadratic polynomial J_t and the decision variables h_w . These constraints are thus already suitable for the SDP.

Enforcing the lower-bound constraint (3d). To apply Corollary 1 to enforce this constraint, we impose additional restrictions. First, we restrict vertex and edge sets \mathcal{X}_v and \mathcal{X}_e to be intersections of ellipsoids and polyhedra. We also restrict edge lengths l_e to be quadratic, ensuring that the expression in (3d) is quadratic. For non-quadratic l_e , such as the Euclidean distance, we use a quadratic approximation instead. Applying Corollary 1, we verify constraint (3d) with an LMI.

The objective function (3a). Since J_s is a quadratic polynomial, the integral in (3a) is linear in the coefficients of J_s , which are the decision variables of the program. Therefore, the objective function (3a) is linear in the decision variables, as required for the SDP.

Empirically, we found quadratic lower bounds to be a good balance between computational complexity and expressive power. Note that higher-degree polynomial lower bounds J_v can be synthesized via the Sums-of-Squares (SOS) hierarchy [27,28,20]. However, in practice, the resulting programs tend to be prohibitively expensive. On the other hand, restricting J_v to be affine yields a program that almost exactly matches the dual to the convex relaxation of the SPP in GCS, discussed in [26, App. B]. In other words, solving the SPP in GCS already gives a coarse affine cost-to-go lower bound that can be used to solve the APSP in GCS. In Section 5.3, we show that empirically, these affine lower bounds have significantly less expressive power than the quadratic lower bounds.

4 Online phase: greedy multi-step lookahead policy

We now generalize the greedy successor policy (1) from the classical APSP to the GCS setting. Suppose that at runtime, we are given a source vertex $v_0 \in \mathcal{S}$ and a source point $x_0 \in \mathcal{X}_{v_0}$. At iteration k of the policy rollout, let (v_k, x_k) be the current vertex and vertex variable, and let $p_k = (v_0, v_1, \dots, v_{k-1})$ be the path so far. The successor policy $\pi(v_k, x_k, p_k) = (v_{k+1}, x_{k+1})$, which we will define shortly, produces the next vertex v_{k+1} and the corresponding vertex variable x_{k+1} . We then advance to the next iteration. The rollout terminates when we reach the target vertex t , where we must select the target point x_t . Upon termination, we extract the vertex path $p = (v_0, v_1, \dots, v_t)$ and re-optimize for the continuous vertex variables (x_0, x_1, \dots, x_t) , so as to produce a trajectory that is optimal within this path.

At each iteration of the policy rollout, we solve a greedy lookahead optimization problem with the coarse quadratic lower bounds obtained in Section 3.2. For simplicity, here we present just the 1-step lookahead program:

$$\pi(v_k, x_k, p_k) = \arg \min_{(w, x_w)} l_e(x_k, x_w) + J_w(x_w) \quad (7a)$$

$$\text{s.t. } e = (v_k, w) \in \mathcal{E}, \quad w \notin p_k, \quad (7b)$$

$$(x_k, x_w) \in \mathcal{X}_e. \quad (7c)$$

Note that we do not use the penalty-incremented edge cost $l_e(x_k, x_w) + h_w$ in (7a), since the penalty h_w is waived the first time that w is visited. Vertex revisits are then also explicitly prohibited in (7b).

In a multi-step lookahead formulation, we instead solve for an n -step optimal decision sequence, take just the first step, and repeat at next iteration. The multi-step lookahead is key for mitigating the coarseness of the quadratic lower bounds. This is because an n -step lookahead from vertex v effectively produces

a piecewise-quadratic lower bound on the cost-to-go J_v^* over \mathcal{X}_v , which has significantly more expressive power. While these lower bounds can still be loose in theory, the multi-step lookahead enables effective decision-making in practice.

Convexity of J_w is crucial, as it allows us to solve the program (7) efficiently at run-time. To find the minimizer to (7), we solve multiple convex programs in parallel, one for every n -step lookahead sequence.

Finally, we note that the lookahead program (7) is not guaranteed to be recursively feasible. If we end up in a vertex where (7) has no solution, we backtrack to a previous vertex that has a different feasible outgoing edge, and retry from there. Generally, our planner is sound but not complete: it is not guaranteed to produce a solution, but every solution it produces is feasible.

5 Experimental evaluation

We evaluate our approach through multiple numerical experiments. Section 5.1 presents a simple two-dimensional problem that provides visual intuition to our method. In Section 5.2, we apply our approach to a complex high-dimensional scenario, the robot arm in Fig. 1. Finally, Section 5.3 shows that our approach scales well to large graphs. We also discuss how the coarseness of the cost-to-go lower bounds and the multi-step lookahead horizon impact the performance.

All of the experiments are run on a desktop computer with a 4.5Ghz 16-core AMD Ryzen 9 processor and 64GB 4800MHz DDR5 memory. We use Mosek 10.2.1 [1] to solve all the convex programs in this section.

5.1 Two-dimensional example

We first consider a two-dimensional GCS problem in Fig. 2. We have a graph G with $|\mathcal{V}| = 9$ vertices, $|\mathcal{E}| = 25$ edges, including multiple cycles. The geometry of the convex sets \mathcal{X}_v can be deduced from Fig. 2a; no edge constraints \mathcal{X}_e are used. The edge costs $l_e(x_v, x_w) = \|x_v - x_w\|_2^2$ are the squared Euclidean distance. The source vertex s is a box, and the target vertex t is a singleton.

We compute the convex quadratic lower bounds on the cost-to-go function at every vertex and visualize their contour plots in Fig. 2a. In Fig. 2b, we depict the first three iterations of the 1-step lookahead rollout of the successor policy (7). At each iteration, we expand the neighbours of the current vertex and greedily select the next vertex w and the vertex point x_w that minimize the objective (7a). The rollout proceeds until the target vertex t is reached.

We evaluate the quality of the cost-to-go lower bounds and the resulting solutions in Fig. 4. The optimal shortest path cost-to-go function J_s^* (green) is piecewise-quadratic. Naturally, the convex quadratic lower bound J_s (purple) is a poor lower bound to J_s^* . The quality of the lower bound is greatly improved via multi-step lookaheads (solid lines, orange for 1-step, blue for 2-step). A horizon- n lookahead produces a piecewise-quadratic lower bound to J_s^* , with up to as many quadratic pieces as there are different n -step paths from the source vertex s . Though neither 1-step nor 2-step lookahead lower bounds are tight, they

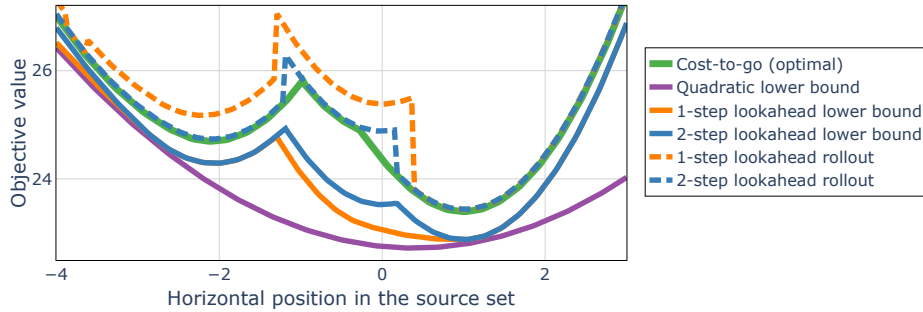


Fig. 4: Comparison of lower and upper bounds on the cost-to-go over a horizontal slice of the source set \mathcal{X}_s from Fig. 2a. The cost-to-go function J_s^* (green) is piecewise-quadratic. Convex quadratic lower bound J_s (purple) is naturally a poor lower-bound. Multi-step lookaheads (solid orange, blue) produce tighter piecewise-quadratic lower bounds. Upper bounds on the cost-to-go are obtained by rolling out the multi-step lookahead policy (dashed orange, blue), which produces near-optimal solutions.

are sufficient for near-optimal decision making. The costs of the rollouts of the successor policy are plotted as dashed lines; 2-step lookahead rollouts (blue) attain optimal solutions nearly always.

5.2 Collision-free motion planning for a robot arm

We now demonstrate that our approach scales well to high-dimensional hardware systems. We study multi-query collision-free motion planning for the KUKA iiwa robotic arm (Fig. 1), tasked with moving virtual items between shelves and bins. Our methodology requires minimal additional offline computation, while delivering significant online speed up with negligible solution quality reduction.

We first produce an approximate polytopic decomposition of the 7-dimensional collision-free configuration space of the arm. This is done via the IRIS-NP algorithm [29], and we use IRIS clique seeding [36] to obtain polytopes inside the shelves and bins. We assign a GCS vertex v per polytope in this decomposition. The convex set \mathcal{X}_v is the set of linear segments contained within the region, with the segment represented by its endpoints. Two GCS vertices are connected by an edge if the corresponding regions overlap. The resulting graph contains 23 vertices and 68 edges. For each edge $e = (v, w)$, we constrain the linear segments at v and w to form a continuous path. The path length is the sum of the Euclidean distances of the linear segments. We define 12 source vertices (6 shelves, 2 vertices per shelf) and 3 target vertices (inside the left, front, and right bins). To generate the quadratic lower bounds on the cost-to-go function, we use the generalization of (3) discussed in Appendix A.

We evaluate our algorithm in a multi-query scenario: at runtime, the arm is given a random next position to go to, alternating between shelves and bins. We

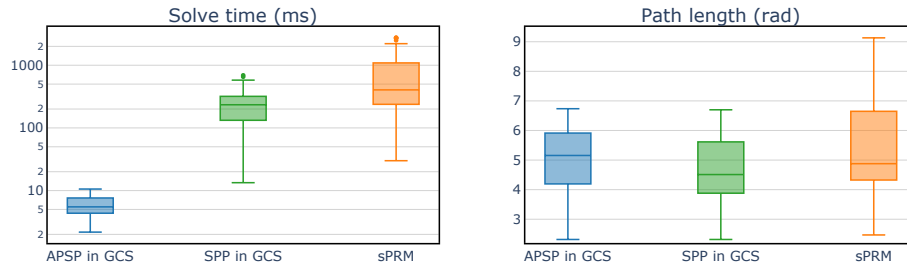


Fig. 5: For the robot arm scenario in Section 5.2, we compare path length and solve time performance between the APSP in GCS, single-query SPP in GCS, and shortcut PRM over 120 queries. The offline phases take 106s, 100s, and 0.9s respectively. The APSP in GCS is on average 40 times faster than the SPP in GCS, with minimal reduction in solution quality. Compared to sPRM, the APSP in GCS is on average 110 times faster.

rollout a 1-step lookahead policy to generate paths from shelves to bins, and reverse them to obtain paths from bins to shelves. We evaluate our approach on a total of 120 queries. We compare our algorithm against solving the SPP in GCS from scratch, as well as against the shortcut PRM (sPRM) algorithm, which is its natural sampling based multi-query competitor. We use a high-performance implementation of sPRM based on [31], producing a large roadmap with 10,000 vertices. Our solutions are visualized in Fig. 1; performance comparison is provided in Fig. 5. Similar to how the quality of the PRM solutions depends on the density of the PRM, the quality of solutions obtained with GCS depends on the quality of the polytopic decomposition of the collision-free configuration space. We thus make no claims about the optimality of the solutions in this section.

Offline, generating cost-to-go lower bounds takes only 6 seconds, which is just 6% of the time that it takes to generate the polytopic decomposition necessary to use GCS. Then online, our policy rollouts are very fast, with a median solve time of 5ms and a maximum of 11ms (we report the parallelized solver time). Our method is on average 40 times faster than the SPP in GCS, producing paths that are only 7% longer on average. Compared to sPRM, our method is on average 110 times faster and produces paths that are 5% shorter on average. We achieve consistent performance in both solve time and path length, unlike sPRM, which shows high variance in both. Overall, compared to these state-of-the-art baselines, the APSP in GCS reduces the online solve times significantly, with minimal compromise in solution quality.

5.3 Scalability and ablation on lower bounds and lookahead horizon

In this section, we demonstrate the scalability of our approach and analyze how the coarseness of the cost-to-go lower bounds and the lookahead horizon impact solution quality. First, we show that multi-step lookaheads with quadratic J_v yield near-optimal solutions in large graphs. Second, we demonstrate that

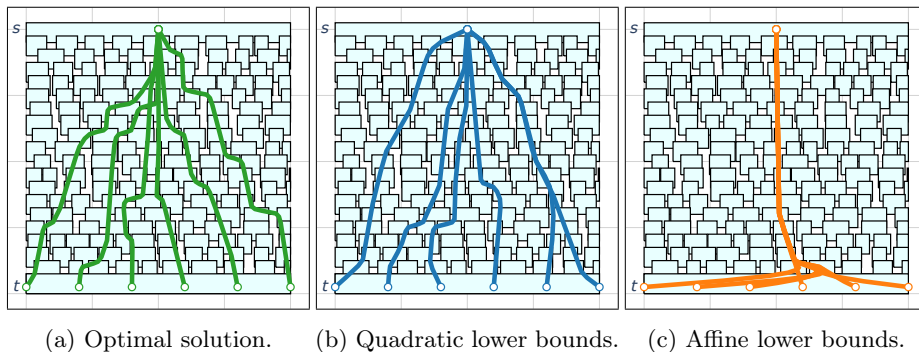


Fig. 6: A 3-step lookahead policy with quadratic J_v (blue) yields diverse vertex paths resembling the optimal solutions (green). A 3-step lookahead with affine J_v (orange) follows a single vertex sequence regardless of the target point, accruing much higher cost.

quadratic lower bounds significantly outperform the affine ones, which are available from the dual of the convex relaxation of the SPP in GCS [26, App. B].

We consider a randomly generated environment depicted in Fig. 6. We assign a GCS vertex v for each teal box. Each convex set \mathcal{X}_v is the set of control points of a cubic Bézier curve within the box (see [25]). The GCS vertices are connected by a pair of edges if the corresponding teal boxes overlap. The resulting graph has 190 vertices and 540 edges. For each edge, we constrain the vertex Bézier curves to be differentiable at the transition point. The path cost is the sum of squared Euclidean distances between the consecutive control points of the Bézier curves. The source vertex s is at the top, and the target vertex t is at the bottom.

We synthesize the quadratic and affine lower bounds over the GCS, which takes 6s and 2s respectively. We then uniformly sample 120 pairs of source and target conditions, and rollout the greedy policy using different lower bounds and lookahead horizons. Optimal solutions are obtained by solving the MICP formulation of the SPP in GCS. Numerical results are reported in Table 1.

Table 1 shows that our approach scales well to large problem instances, yielding better solve times than the SPP in GCS. A 2-3 step lookahead policy with a quadratic cost-to-go lower bound produces near-optimal solutions (8-9% median suboptimality) in under 10ms. The SPP in GCS produces slightly better solutions (7% median suboptimality), but due to the size of the graph, the solve-time increases to over 1000ms. For large graph instances, incremental search through the graph via the APSP in GCS achieves competitive solution quality while reducing solve times by up to two-three orders of magnitude.

Finally, Table 1 shows that quadratic lower bounds with short-horizon lookaheads offer a good balance between expressive power and solve times. A 3-step lookahead policy with affine lower bounds has a median suboptimality of 80.2%, compared to 8.8% with quadratic lower bounds. Achieving similar solution quality with affine lower bounds requires a lookahead horizon of 8-9 steps, but the

Solution method	Optimality gap, %	Solve time, ms	Failure rate, %
Quadratic J_v , 1-step	20.0 (62.1)	3 (3)	0.0
Quadratic J_v , 2-step	9.4 (22.3)	4 (4)	0.0
Quadratic J_v , 3-step	8.8 (15.7)	5 (6)	0.0
Affine J_v , 1-step	157.1 (N/A)	2 (657)	27.2
Affine J_v , 2-step	142.4 (418.8)	3 (914)	14.0
Affine J_v , 3-step	80.2 (348.3)	5 (808)	9.9
Affine J_v , 8-step	11.9 (37.4)	169 (1996)	3.3
Affine J_v , 9-step	7.0 (26.2)	388 (2454)	0.0
SPP in GCS	6.9 (12.0)	716 (1051)	0.0

Table 1: Impact of the degree of J_v and lookahead horizon on performance, over 120 queries for the GCS in Fig. 6. We report optimality gaps (ratio between solution cost and optimal cost), solve times, and failure rates (rollout policy is terminated after 10,000 iterations). We report median values, with the 75th percentile in the parenthesis. Low-horizon lookahead policies with quadratic lower bounds yield near optimal solutions, perform much better than the affine bounds.

resulting rollouts take significantly more time. Fig. 6 shows that 3-step lookahead rollouts with affine lower bounds fail to capture the diversity of optimal solutions. Additionally, low-horizon lookahead policies with affine lower bounds often fail to produce solutions within a reasonable number of iterations, as demonstrated by the failure rate statistics. Overall, we observe that the lookahead policies with quadratic lower bounds perform much better than those with affine ones.

6 Conclusion and future work

In this work, we generalized the classical All-Pairs Shortest-Paths problem to the Graphs of Convex Sets, and developed practical approximate numerical methods for solving this problem. We demonstrated that a coarse lower bound on the cost-to-go with a greedy multi-step lookahead policy produce near-optimal paths, while significantly reducing solve times. Our methodology effectively scales to high-dimensional set scenarios and large graph instances, enabling practical robotics applications in multi-query settings. We plan to provide an efficient implementation of our approach within the Drake library [33].

For hardware applications in non-static environments, we are interested in ways to tackle changes to the robot’s configuration space, like those arising in object manipulation, as well as addition and removal of obstacles. Assuming the changes are minor, the online search via the multi-step lookahead policy provides natural local adaptation. Changes to the environment can be incorporated into the online policy rollout program (7) via non-convex constraints, similar to [37].

Finally, we are interested in exploring alternative incremental search policies beyond the multi-step lookahead policy. We expect randomized rollouts inspired by MCTS [7] and A*-based approaches like [8] to be effective.

References

1. ApS, M.: The MOSEK optimization toolbox for MATLAB manual. Version 10.1. (2024), <http://docs.mosek.com/latest/toolbox/index.html>
2. Bellman, R.: Dynamic programming. *science* **153**(3731), 34–37 (1966)
3. Bemporad, A., Morari, M., Dua, V., Pistikopoulos, E.N.: The explicit linear quadratic regulator for constrained systems. *Automatica* **38**(1), 3–20 (2002)
4. Bertsekas, D.: *Dynamic programming and optimal control*, vol. 4. Athena scientific (2012)
5. Blekherman, G., Parrilo, P.A., Thomas, R.R.: *Semidefinite optimization and convex algebraic geometry*. SIAM (2012)
6. Bohlin, R., Kavraki, L.E.: Path planning using lazy PRM. In: *Proceedings 2000 ICRA. Millennium conference. IEEE international conference on robotics and automation. Symposia proceedings (Cat. No. 00CH37065)*. vol. 1, pp. 521–528. IEEE (2000)
7. Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games* **4**(1), 1–43 (2012)
8. Chia, S.Y.C., Jiang, R.H., Graesdal, B.P., Kaelbling, L.P., Tedrake, R.: GCS*: Forward heuristic search on implicit graphs of convex sets. *arXiv preprint arXiv:2407.08848* (2024)
9. Cohn, T., Petersen, M., Simchowicz, M., Tedrake, R.: Non-Euclidean motion planning with graphs of geodesically-convex sets. *Robotics: Science and Systems* (2023)
10. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: *Introduction to algorithms*. MIT press (2022)
11. De Farias, D.P., Van Roy, B.: The linear programming approach to approximate dynamic programming. *Operations research* **51**(6), 850–865 (2003)
12. Floyd, R.W.: Algorithm 97: shortest path. *Communications of the ACM* **5**(6), 345–345 (1962)
13. Graesdal, B.P., Chia, S.Y., Marcucci, T., Morozov, S., Amice, A., Parrilo, P.A., Tedrake, R.: Towards tight convex relaxations for contact-rich manipulation. *Robotics: Science and Systems* (2024)
14. Jaillet, L., Siméon, T.: A PRM-based motion planner for dynamically changing environments. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*(IEEE Cat. No. 04CH37566). vol. 2, pp. 1606–1611. IEEE (2004)
15. Johnson, D.B.: Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM (JACM)* **24**(1), 1–13 (1977)
16. Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. *The international journal of robotics research* **30**(7), 846–894 (2011)
17. Kavraki, L.E., Svestka, P., Latombe, J.C., Overmars, M.H.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation* **12**(4), 566–580 (1996)
18. Kuffner, J.J., LaValle, S.M.: RRT-connect: An efficient approach to single-query path planning. In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*. vol. 2, pp. 995–1001. IEEE (2000)
19. Kurtz, V., Lin, H.: Temporal logic motion planning with convex optimization via graphs of convex sets. *IEEE Transactions on Robotics* **39**(5), 3791–3804 (2023)

20. Lasserre, J.B.: Global optimization with polynomials and the problem of moments. *SIAM Journal on optimization* **11**(3), 796–817 (2001)
21. Lasserre, J.B., Henrion, D., Prieur, C., Trélat, E.: Nonlinear optimal control via occupation measures and LMI-relaxations. *SIAM journal on control and optimization* **47**(4) (2008)
22. LaValle, S.: Rapidly-exploring random trees: A new tool for path planning. Research Report 9811 (1998)
23. Lewis, F.L., Liu, D.: Reinforcement learning and approximate dynamic programming for feedback control. John Wiley & Sons (2013)
24. Marcucci, T.: Graphs of Convex Sets with Applications to Optimal Control and Motion Planning. Ph.D. thesis, Massachusetts Institute of Technology (2024)
25. Marcucci, T., Petersen, M., von Wrangel, D., Tedrake, R.: Motion planning around obstacles with convex optimization. *Science robotics* **8**(84), eadf7843 (2023)
26. Marcucci, T., Umenberger, J., Parrilo, P., Tedrake, R.: Shortest paths in graphs of convex sets. *SIAM Journal on Optimization* **34**(1), 507–532 (2024)
27. Parrilo, P.A.: Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization. California Institute of Technology (2000)
28. Parrilo, P.A.: Semidefinite programming relaxations for semialgebraic problems. *Mathematical programming* **96**, 293–320 (2003)
29. Petersen, M., Tedrake, R.: Growing convex collision-free regions in configuration space using nonlinear programming. arXiv preprint arXiv:2303.14737 (2023)
30. Philip, A.G., Ren, Z., Rathinam, S., Choset, H.: A mixed-integer conic program for the moving-target traveling salesman problem based on a graph of convex sets. arXiv preprint arXiv:2403.04917 (2024)
31. Phillips-Grafflin, C.: Common robotics utilities https://github.com/ToyotaResearchInstitute/common_robotics_utilities
32. Powell, W.B.: Approximate Dynamic Programming: Solving the curses of dimensionality, vol. 703. John Wiley & Sons (2007)
33. Tedrake, R., the Drake Development Team: Drake: Model-based design and verification for robotics (2019), <https://drake.mit.edu>
34. Wang, Y., O’Donoghue, B., Boyd, S.: Approximate dynamic programming via iterated Bellman inequalities. *International Journal of Robust and Nonlinear Control* **25**(10) (2015)
35. Warshall, S.: A theorem on boolean matrices. *Journal of the ACM (JACM)* **9**(1), 11–12 (1962)
36. Werner, P., Amice, A., Marcucci, T., Rus, D., Tedrake, R.: Approximating robot configuration spaces with few convex sets using clique covers of visibility graphs. *International Conference on Robotics and Automation* (2024)
37. von Wrangel, D., Tedrake, R.: Using graphs of convex sets to guide nonconvex trajectory optimization

A Extensions and variations

We briefly remark on various natural generalizations to program (3).

1. Suppose the set of source vertices \mathcal{S} has more than one vertex. To simultaneously “push up” lower bounds J_s per vertex $s \in \mathcal{S}$, we add extra integral terms to the objective function (3a).
2. Suppose the target set \mathcal{X}_t is not a singleton, but a compact convex set. First, we modify the constraint (3b) to search for $J_{v,t} : \mathcal{X}_v \times \mathcal{X}_t \rightarrow \mathbb{R}$. The function $J_{v,t}(x_v, x_t)$ is a lower bound on the cost-to-go of the shortest path from x_v of vertex v to x_t of vertex t . Similarly, the probability distribution $\phi_{s,t}$ is now supported on $\mathcal{X}_s \times \mathcal{X}_t$, so as to push up on $J_{s,t}(x_s, x_t)$ over all source-target pairs (x_s, x_t) . The lower-bound constraint (3d) is adjusted to include $x_t \in \mathcal{X}_t$:

$$J_{v,t}(x_v, x_t) \leq l_e(x_v, x_w) + h_w + J_{w,t}(x_w, x_t),$$

for all edges $e = (v, w) \in \mathcal{E}$, and all points $(x_v, x_w) \in \mathcal{X}_e$ and $x_t \in \mathcal{X}_t$. Finally, the target constraint (3e) is adjusted to be $J_{t,t}(x_t, x_t) = -\sum_{w \in \mathcal{V}} h_w$, for all $x_t \in \mathcal{X}_t$.

3. The scalar vertex penalty h_w is generalized to be a non-negative function of the target state x_t , that is: $h_{w,t} : \mathcal{X}_t \rightarrow \mathbb{R}_+$. We thus replace h_w with $h_{w,t}(x_t)$ and update the constraint (3e) as follows:

$$J_{t,t}(x_t, x_t) = -\sum_{w \in \mathcal{V}} h_w(x_t),$$

further tightening the resulting lower-bounds.

4. Suppose the set of target vertices \mathcal{T} has more than one vertex. To obtain the cost-to-go lower bounds for every pair of vertices $v \in \mathcal{V}$ and $t \in \mathcal{T}$, we solve multiple programs (3) in parallel, one per target vertex $t \in \mathcal{T}$.
5. In general, the successor policy (7) is also a function of terminal vertex t and terminal point x_t . The generalized 1-step lookahead program is as follows:

$$\begin{aligned} \pi(v_k, x_k, p_k, t, x_t) &= \arg \min_{(w, x_w)} l_e(x_k, x_w) + J_{w,t}(x_w, x_t) \\ \text{s.t.} \quad &e = (v_k, w) \in \mathcal{E}_{v_k}^{\text{out}}, \quad w \notin p_k, \\ &(x_k, x_w) \in \mathcal{X}_e. \end{aligned}$$

6. Other penalties, similar to the vertex visitation penalties h_v , can be added to improve the quality of the lower bounds. For instance, consider a 2-cycle with edges (v, w) and (w, v) . We can add edge penalties $h_{v,w} = h_{w,v}$ for traversing either edge. By subtracting $h_{v,w}$ from the cost-to-go lower bound at the target, we effectively ensure that no penalty is incurred for traversing just one (but not both) of the edges. This can be extended to cycles of arbitrary length.