Mixed Discrete and Continuous Planning using Shortest Walks in Graphs of Convex Sets

Savva Morozov[1], Tobia Marcucci[2], Bernhard Paus Graesdal[1], Alexandre Amice[1], Pablo A. Parrilo[1], and Russ Tedrake[1]

[1] Massachusetts Institute of Technology
{savva, graesdal, amice, parrilo, russt}@mit.edu
[2] University of California, Santa Barbara
marcucci@ucsb.edu

Abstract. This paper introduces the Shortest-Walk Problem (SWP) in Graphs of Convex Sets (GCS). A GCS is a graph where vertices are paired with convex programs, and edges couple adjacent vertex programs through additional convex costs and constraints. A walk in a GCS is a sequence of potentially repeated vertices connected by edges; its cost is given by the sum of its vertex and edge costs. To solve the SWP in GCS, we first synthesize a piecewise-quadratic lower bound on the problem's cost-to-go function using semidefinite programming. Then we use this lower bound to guide an incremental search that yields an approximate shortest walk. We show that the SWP in GCS is a natural language for a variety of mixed discrete-continuous planning problems in robotics and control, unifying problems that typically require specialized solution methods while retaining computational efficiency.

1 Introduction

A Graph of Convex Sets (GCS) is a generalization of a weighted graph where each vertex is paired with a convex program, and each edge couples adjacent programs through additional convex costs and constraints [35]. When traversing a GCS, we must select a feasible point for the program associated with each visited vertex, and ensure that consecutive points satisfy the constraints associated with the traversed edges. The total traversal cost is the sum of the costs of these vertices and edges.

Many classical problems in graph theory extend naturally to a GCS [35]. Among these, the Shortest-Path Problem (SPP) in GCS [39] has received particular attention for its many applications and solution efficiency. The SPP in GCS requires optimizing a discrete path through the graph and the associated continuous vertex points jointly. It naturally models problems where graph search and convex optimization are interleaved, and is a powerful tool for optimal control of hybrid systems [39], collision-free motion planning [38, 9], planning through contact [19], and other robotics problems [49, 29].

In this paper, we study the Shortest-Walk Problem (SWP) in GCS: instead of seeking a path through the GCS, which is a sequence of distinct vertices, we seek a walk, which allows vertex revisits. For ordinary weighted graphs with non-negative scalar weights, the SWP reduces to the SPP, since vertex revisits incur non-negative cost without advancing towards the target. In contrast, the SWP and the SPP differ substantially

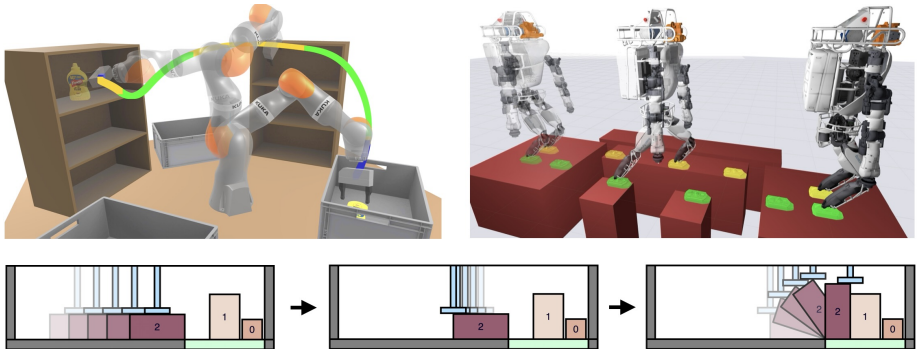


Fig. 1: Application of the SWP in GCS to different planning problems: collision-free motion planning for a robot arm with derivative constraints, footstep planning for a humanoid robot, and item sorting for a top-down suction gripper.

in a GCS, since vertex revisits still incur non-negative cost but allow progress towards the target. We show that the SWP in GCS captures a wide range of robotics and control problems more naturally than the SPP in GCS, including collision-free motion planning, skill chaining, and optimal control of hybrid dynamical systems. Figure 1 highlights instances of these problems analyzed in our experiments.

To solve the SWP in GCS, we extend the method proposed in [43] for the SPP in GCS. First, we synthesize a piecewise-quadratic lower bound on the problem cost-to-go function, with each quadratic piece defined over the convex set of a vertex. Next, we use this bound to guide a search algorithm, obtaining a walk one vertex at a time.

Paper outline. In Section 2, we formulate the SWP in GCS, highlighting its key differences from the SPP in GCS. In Section 3, we demonstrate the relevance of the SWP in GCS in robotics and optimal control through instances where repeated node visits arise naturally, and review the literature on solution techniques for discrete–continuous problems. Section 4 describes a practical numerical approach for solving the SWP in GCS. We collect multiple numerical experiments in Section 5 and conclude in Section 6.

2 Shortest walks in graphs of convex sets

In this section, we review some background material, state the SWP in GCS, and highlight its main differences with respect to the SPP in GCS.

2.1 Definitions

A GCS generalizes a weighted graph by allowing vertex and edge weights (or costs) to depend on continuous variables. Formally, let $G = (\mathcal{V}, \mathcal{E})$ be a directed graph with vertex set $\mathcal{V}$ and edge set $\mathcal{E}$. Each vertex $v \in \mathcal{V}$ is associated with a convex program, specified by a compact convex set $\mathcal{X}_v$ and a non-negative convex cost function $l_v : \mathcal{X}_v \to \mathbb{R}_+$. Similarly, each edge $e = (u, v) \in \mathcal{E}$ is associated with a convex set $\mathcal{X}_e \subseteq$

$\mathcal{X}_u \times \mathcal{X}_v$ and a non-negative convex cost function $l_e : \mathcal{X}_e \to \mathbb{R}_+$. When traversing a GCS, visiting a vertex v requires selecting a point $x_v \in \mathcal{X}_v$ and incurs the cost $l_v(x_v)$. Moving along an edge $e = (u, v)$ requires the adjacent points x_u and x_v to satisfy $(x_u, x_v) \in \mathcal{X}_e$ and incurs the cost $l_e(x_u, x_v)$.

Let $s \in \mathcal{V}$ and $t \in \mathcal{V}$ be a pair of source and target vertices, and $\bar{x}_s \in \mathcal{X}_s$ and $\bar{x}_t \in \mathcal{X}_t$ be a pair of source and target points. A *K-step walk in a GCS* between vertices s and t, and points $\bar{x}_s$ and $\bar{x}_t$, is a sequence of $K+1$ vertices $w = (v_0, \ldots, v_K)$ and points $\tau = (x_0, \ldots, x_K)$ such that

$$v_0 = s,\ v_K = t, \tag{1a}$$
$$x_0 = \bar{x}_s,\ x_K = \bar{x}_t, \tag{1b}$$
$$e_k = (v_{k-1}, v_k) \in \mathcal{E}, \qquad \forall k = 1, \ldots, K, \tag{1c}$$
$$(x_{k-1}, x_k) \in \mathcal{X}_{e_k}, \qquad \forall k = 1, \ldots, K. \tag{1d}$$

In words, the walk starts at vertex s and point $\bar{x}_s$, and ends at vertex t and point $\bar{x}_t$. Consecutive vertex pairs (v_{k-1}, v_k) are connected by an edge $e_k \in \mathcal{E}$, and consecutive point pairs (x_{k-1}, x_k) lie in the edge constraint set $\mathcal{X}_{e_k}$. The latter also ensures that the vertex constraints are satisfied along the walk, since $\mathcal{X}_{e_k} \subseteq \mathcal{X}_{v_{k-1}} \times \mathcal{X}_{v_k}$ by assumption.

The tuple (w, τ) is the walk in the GCS. Individually, the sequence of vertices w is a walk in the graph G, while τ is the corresponding sequence of vertex points, referred to as the *trajectory*. We denote by $\mathcal{W}_{s,t}(K)$ the set of K-step walks w that satisfy (1a) and (1c). We denote by $\mathcal{T}_{\bar{x}_s, \bar{x}_t}(w)$ the set of trajectories τ along the walk w that satisfy (1b) and (1d). We emphasize that vertices and edges along the walk w may be repeated, and that different continuous points may be selected upon revisiting the same vertex.

The cost of the walk is the sum of its edge costs and all but the first vertex costs:

$$l(w, \tau) = \sum_{k=1}^{K} \Big(l_{e_k}(x_{k-1}, x_k) + l_{v_k}(x_k) \Big).$$

2.2 Problem statement

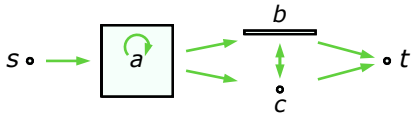
The shortest walk in the GCS is the solution to the following optimization problem:

$$\inf_{K} \min_{w,\tau} \quad l(w, \tau) \tag{2a}$$
$$\text{s.t.} \quad w \in \mathcal{W}_{s,t}(K), \quad \tau \in \mathcal{T}_{\bar{x}_s, \bar{x}_t}(w). \tag{2b}$$

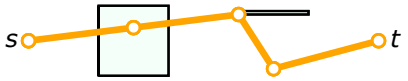
Note that this is a bilevel optimization: we seek the shortest K-step walk (w, τ) at the lower level, and take the infimum over K at the higher level. We thus optimize over the number of steps K, the walk w through the graph, and the trajectory τ along this walk.

2.3 Shortest walks need not be paths

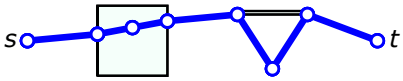
For an ordinary graph with non-negative edge costs, the SWP always admits an optimal solution that is a path (i.e., a sequence of vertices with no repetitions). This is because revisiting a vertex creates a cycle of non-negative cost and makes no progress towards



(a) GCS embedded in $\mathbb{R}^2$.



(b) Shortest path between vertices s and t.



(c) Shortest walk between vertices s and t.

Fig. 2: Example of a shortest walk that is not a path. Vertices a and b are both revisited along the walk in panel (c): a is visited three times consecutively, and b is visited twice non-consecutively.

the target. Therefore, this cycle may be removed without increasing the cost of the walk. The same is not true for walks in a GCS. When traversing a GCS, revisiting the same vertex can be advantageous, as demonstrated by the following example.

Example 1. Consider the 2D problem depicted in Figure 2a. This GCS has 5 vertices $\mathcal{V} = \{s, a, b, c, t\}$ and 8 edges, drawn in green. The convex set $\mathcal{X}_a$ is a square, $\mathcal{X}_b$ is a segment, and $\mathcal{X}_s, \mathcal{X}_c, \mathcal{X}_t$ are points. For every edge $e = (u, v)$, the edge cost is defined as $l_e(x_u, x_v) = 1 + ||x_u - x_v||_2^2$: the first term penalizes the number of steps taken, while the squared displacement term penalizes the size of each step. There are no vertex costs, nor are there any additional edge constraints. The solutions to the SPP and the SWP in this GCS are shown in Figures 2b and 2c. To avoid revisiting vertices, the shortest path (orange) must take larger steps, incurring a cost of 35. By taking smaller steps and revisiting vertices, the shortest walk (blue) achieves a lower cost of 28. Although the cycles in the GCS have a non-negative cost, they help make progress towards the target.

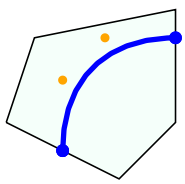
2.4 Sufficient condition for finiteness of shortest walks

We note that the shortest walk in a GCS need not be of finite length K. Consider again the GCS in Figure 2a. If the edge cost is set to $l_e(x_u, x_v) = ||x_u - x_v||_2^2$, then the optimal walk involves infinitely-many infinitesimal steps through the vertex a. No finite walk is optimal, as there is always a longer walk that attains a lower cost. The infimum cost is finite, but not attained: it is approached in the limit of walks with infinitely many steps. That is why we defined program (2) using an infimum over the number of steps K.

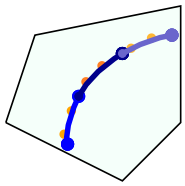
Below is a simple sufficient condition for an optimal walk, if one exists, to be finite:

$$\min\{l_e(x_u, x_v) + l_v(x_v) \mid (x_u, x_v) \in \mathcal{X}_e\} > 0, \qquad \forall e = (u, v) \in \mathcal{E}.$$

This condition ensures that the cost of every step in the walk is greater than some positive value; each step cannot have arbitrarily small cost, unlike the example in the last paragraph. Thus an infinite walk must incur infinite cost, and cannot be optimal. In practice, this condition can be satisfied by adding a small constant $\epsilon > 0$ to every edge cost l_e. In what follows, we assume that this condition holds.



(a) The shortest-path formulation from [38] optimizes a Bézier curve and its duration, resulting in non-convex acceleration constraints.



(b) Our shortest-walk formulation searches for a fixed-duration Bézier curve per vertex visit, resulting in convex acceleration constraints.

Fig. 3: Comparison of shortest-path and shortest-walk formulations for motion planning. Shown in blue are cubic Bézier curves, with their control points in orange.

3 Applications in robotics and control

In this section, we present three representative problems in robotics and optimal control that are naturally formulated as an SWP in GCS. Although some of these problems have previously been addressed using a shortest-path formulation, we argue that the shortest-walk formulation is more natural and compact. We will also revisit these same problems in Section 5 to show that the SWP formulation is computationally more efficient.

3.1 Collision-free motion planning with acceleration limits

Collision-free motion planning is a key challenge in robotics. Sampling-based planners [26, 31, 13] are a popular approach to this problem due to their simplicity and effectiveness. They offer probabilistic guarantees of feasibility and optimality [25, 22] but, even using their kinodynamic variants [66, 18, 70], handling continuous derivative constraints remains challenging for these methods. Optimization methods [4, 2, 52, 34, 72] handle these constraints by enforcing them in a non-convex program, but often fail to find a feasible solution in complex environments due to their reliance on local solvers. Planners based on mixed-integer optimization [51, 50, 41, 12] combine discrete and continuous search, and find globally optimal trajectories using branch-and-bound solvers. The mixed-integer planner proposed in [38] reformulates the problem as an SPP in GCS. This yields fast solve times but imposes strict convexity requirements on the admissible costs and constraints involving the trajectory derivatives.

The approach in [38] decomposes the collision-free space into convex regions. A GCS vertex is added for every such region, and two vertices are connected by an edge if the corresponding regions overlap. The convex set $\mathcal{X}_v$ that is actually paired with each vertex $v \in \mathcal{V}$ is the set of Bézier curves contained within the corresponding collision-free region, together with the curve time duration. Therefore, visiting a vertex requires selecting a trajectory through a collision-free region, as shown in Figure 3a. A vertex cost on the trajectory velocity and duration is also added. Traversing an edge imposes constraints that guarantee a continuous and differentiable trajectory. A path in this GCS corresponds to a smooth collision-free trajectory from start to target.

A key limitation of this formulation is that jointly optimizing the shape and duration of the Bézier curves results in non-convex constraints on the trajectory acceleration and

higher-order derivatives. One way to address this issue is to relax the non-convex timing or higher-order continuity constraints, and then post-process the resulting trajectories using Time-Optimal Path Parameterization (TOPP) [62] or by solving a non-convex program on the vertex sequence [63, 69, 40]. However, this decouples the problem into two stages and may lead to suboptimal solutions. Alternatively, the nonconvex vertex constraints can be approximated via semidefinite relaxations [71], however, such relaxations can often be loose or computationally expensive. Finally, derivative constraints can be convexified by fixing the duration of each curve and introducing copies of the GCS vertices. This restricts the time spent in each region to integer multiples of the fixed curve duration, but introduces hyper-parameters (the number of vertex copies) and a trade-off between the time discretization error and the problem size.

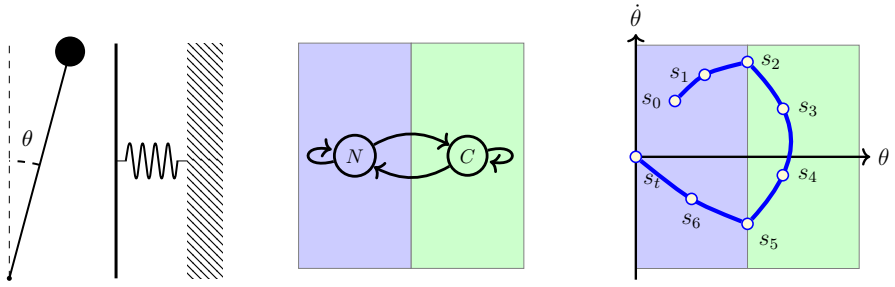
The limitations of the last approach are naturally addressed by the shortest-walk formulation of the motion planning problem. This formulation does not introduce additional hyper-parameters and resolves the trade-off between discretization error and problem size by allowing an arbitrary number of vertex revisits. As a result, vertex duplication is unnecessary, and the problem complexity is not artificially inflated.

3.2 Skill chaining

Consider a robotic system controlled by a discrete set of continuously parameterized skills (also referred to as motion primitives, actions, or behaviors) that use low-level control policies to transition between configurations. Abstracting away the low-level dynamics of these policies, the goal of skill chaining is to select a sequence of skills and the corresponding control parameters that allow the robot to reach a target configuration [58, 27]. Related families of problems include sequential composition [6, 59] and Task and Motion Planning (TAMP, see [16] for a comprehensive review). A common solution strategy alternates between sampling discrete skills and continuous transitions (control parameters), guided by strong heuristics [7, 23, 55, 28, 15]. However, to be effective in complex environments, these methods often rely on costly hand-crafted samplers. Optimization-based subroutines [61, 20, 53, 14] can also be used to explore the space of continuous transitions more effectively and better inform the discrete search. The approach we propose in this subsection for skill chaining builds on this idea.

Let n be the dimension of the configuration space. We define a skill π via a set $\mathcal{Q}_\pi \subset \mathbb{R}^{2n}$ of feasible configuration pairs that can be connected by the skill. Note that alternative definitions in the literature describe skills through preconditions (pre-image, domain, or initiation set) and effects (reachable set, goal set, or termination condition), but all are generally interchangeable. Each skill also has an associated cost function $c_\pi : \mathcal{Q}_\pi \to \mathbb{R}_+$, where $c_\pi(q, q')$ is the cost of transitioning from configuration q to q' under this skill. Given a pair of start and target configurations $\bar{q}_s, \bar{q}_t$, the goal is to find a minimum-cost sequence of skills $(\pi_1, \dots, \pi_K)$, with a corresponding sequence of transitions $\big((q_0, q_1), \dots, (q_{K-1}, q_K)\big)$, such that $q_0 = \bar{q}_s$, $q_K = \bar{q}_t$, and each transition (q_{k-1}, q_k) is achievable via π_k.

To formulate the skill chaining problem as an SWP in GCS, we require that the sets $\mathcal{Q}_\pi$ and the cost functions c_π be convex. If this is not the case, we assume that suitable convex approximations or decompositions are available (existing tools such as [11, 48, 68] can facilitate this process). We let each skill π be a GCS vertex and pair



(a) (b) (c)

Fig. 4: An actuated pendulum with a soft wall (a) is approximated as a PWA system with two modes (no-contact and contact), and modeled as a GCS with two vertices and four edges (b). An optimal state-space trajectory for regulating the pendulum to the equilibrium position $s_t = (0, 0)$ is computed by solving an SWP in GCS (c).

it with the convex set $\mathcal{X}_\pi = \mathcal{Q}_\pi$ and the convex cost $l_\pi = c_\pi$. Visiting vertex π is thus equivalent to executing a transition (q, q') and incurs a vertex cost $c_\pi(q, q')$. An edge connects two skills π_1 and π_2 if they can be chained, i.e., there exist configurations q, q', q'' such that $(q, q') \in \mathcal{Q}_{\pi_1}$ and $(q', q'') \in \mathcal{Q}_{\pi_2}$. Ensuring that the end point q' of the first skill equals the start point of the second skill requires adding an edge constraint. We then add a start vertex for the start configuration $\bar{q}_s$ and connect it to vertices that represent skills executable from $\bar{q}_s$. We add a target vertex in a similar fashion. The optimal skill chaining is then represented by the shortest walk in this GCS.

3.3 Optimal control of hybrid systems

Many challenging problems in robotics, such as footstep planning, planning through contact, and dexterous manipulation, involve systems with hybrid dynamics. Such systems can be well approximated with a Piecewise Affine (PWA) dynamical model [54, 3]. Motivated by this, we consider the problem of optimal control for discrete-time PWA dynamical systems. We refer the reader to [33] for a recent work in this direction, which highlights the SPP in GCS as an effective and competitive strategy. Below we show that the shortest-walk formulation may offer a more effective alternative.

Let $\mathcal{S}$ and $\mathcal{A}$ be the system state and control spaces, and let the state-space be partitioned into polyhedral sets $\mathcal{S} = \bigcup_i \mathcal{S}_i$ (commonly referred to as modes). A PWA system evolves according to different affine dynamics depending on the mode that system is in. That is, the system dynamics at time-step n are governed by:

$$s_{j+1} = A_i s_j + B_i a_j + c_i \quad \text{if} \quad s_j \in \mathcal{S}_i,\ a_j \in \mathcal{A}.$$

Executing the control a_j at state $s_j \in \mathcal{S}_i$ of mode i incurs the mode-specific stage cost $l_i(s_j, a_j)$. We seek state, control, and mode sequences between source and target states $\bar{s}_0$ and $\bar{s}_t$, satisfying the PWA dynamics and minimizing the total stage cost.

This problem can be naturally cast as an SWP in GCS. We illustrate this in Figure 4 using a pendulum with a soft wall: a simple example of control through contact that can

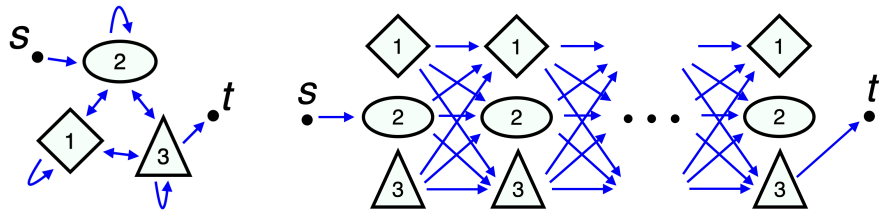


Fig. 5: A simple approach to solving the SWP in GCS in (a) involves constructing the layered GCS in (b) by duplicating vertices across $K-1$ layers, for each $K \to \infty$. In contrast, our approach operates directly on the original, smaller GCS in (a), and avoids artificially increasing the size of the problem.

be effectively approximated as a PWA system [37]. For every mode i, we define a GCS vertex i with convex set $\mathcal{X}_i = \mathcal{S}_i \times \mathcal{A}$. Two vertices are connected with an edge if a feasible transition exists between some pair of states in the corresponding modes. Affine dynamics are imposed as edge constraints, and the convex stage cost l_i is added as a vertex cost. For the pendulum, this results in a GCS with two vertices (C for contact, N for no-contact) and four edges, pictured in Figure 4b. Note that the affine dynamics along the edges (N, C) and (C, N) are different. A walk in this GCS is a vertex sequence w, which corresponds to a mode sequence, and a sequence of points τ, which corresponds to state and control trajectories. This is illustrated in Figure 4c. Note that this PWA control problem can also be cast as an SPP in a GCS, following the construction in [35, §8] and also summarized in Figure 5 below. This formulation yields a GCS with $O(IK)$ vertices and $O(I^2K)$ edges, where I is the number of contact modes and K is the planning horizon, leading to a computationally expensive optimization problem.

4 Solution method

The SWP in GCS is NP-hard, since the SPP in an acyclic GCS is NP-hard [35, §9.2], and the two problems are equivalent in acyclic graphs. Therefore, we do not expect to solve each instance of the SWP in GCS efficiently.

An impractical way to find the shortest walk in a GCS is to compute K-step optimal walks for progressively larger K. As K grows, the cost of the best solution found converges to the infimum in (2). A K-step optimal walk can be obtained by solving an SPP in a different GCS, where we duplicate vertices as a way to allow revisits, as shown in Figure 5. Given the original GCS in Figure 5a, we construct a new layered GCS as in Figure 5b, with each layer containing duplicates of the original vertices, and consecutive layers connected based on the original edges. Solving the SPP in this layered GCS yields a K-step optimal path, equivalent to a K-step optimal walk in the original GCS.

Incremental search offers a more effective approach to the problem. Recent works proposed incremental methods for solving the SPP in GCS [8, 56, 44, 43], which naturally extend to produce walks by allowing vertex revisits. The performance of incre-

mental search depends significantly on the quality of the guiding heuristic: here we develop effective heuristics for the SWP in GCS leveraging the method from [43].

Our approach has two computational steps: (1) obtain cost-to-go lower bounds at each vertex, then (2) use these lower bounds to guide an incremental search that extracts a walk. To compute the lower bounds, we first derive the Bellman equation for the SWP in GCS. This equation can be solved exactly for the cost-to-go via an infinite-dimensional optimization problem; since this is intractable, we approximate it with a semidefinite program over quadratic lower bounds. Notably, this program scales with the size of the original GCS, not the product of graph size and walk length.

4.1 Bellman equation

The principle of optimality holds for the SWP in GCS: every subwalk of a shortest walk is itself a shortest walk. Indeed, if a subwalk was not optimal, then it could be replaced with the actual optimal subwalk, resulting in a walk of lower cost. We leverage this property to derive the Bellman equation for the SWP in GCS.

For every vertex $v \in \mathcal{V}$ and point $x_v \in \mathcal{X}_v$, let $J_v^*(x_v)$ denote the cost of a shortest walk from x_v to the target point $\bar{x}_t$, also referred to as the *cost-to-go*. Consider a point x_u of vertex u, and let the point x_v of vertex v be next along a shortest walk from x_u (vertices u, v need not be distinct). By the principle of optimality, the cost-to-go $J_u^*(x_u)$ must be the sum of the incurred costs $l_e(x_u, x_v) + l_v(x_v)$, with $e = (u, v)$, and the subsequent cost-to-go $J_v^*(x_v)$. As the transition to x_v of vertex v is optimal, it must minimize this sum over all other feasible transitions. This yields the Bellman equation

$$J_u^*(x_u) = \min_{x_v,\, v} \quad l_e(x_u, x_v) + l_v(x_v) + J_v^*(x_v) \tag{3a}$$

$$\text{s.t.} \quad (x_u, x_v) \in \mathcal{X}_e, \quad e = (u, v) \in \mathcal{E}. \tag{3b}$$

4.2 Exact solution of the Bellman equation

We now reformulate the Bellman equation (3) exactly as an infinite-dimensional optimization problem. We search over the functions $J_v : \mathcal{X}_v \to \mathbb{R}_+$ per vertex $v \in \mathcal{V}$ that satisfy the following inequality, which is a relaxed form of the Bellman equation (3):

$$J_u(x_u) \leq l_e(x_u, x_v) + l_v(x_v) + J_v(x_v),$$

for all $e = (u, v) \in \mathcal{E}$ and $(x_u, x_v) \in \mathcal{X}_e$. This inequality states that $J_u(x_u)$ is no greater than the incurred edge and vertex costs $l_e(x_u, x_v) + l_v(x_v)$ plus the subsequent value $J_v(x_v)$. If we constrain $J_t(\bar{x}_t) = l_t(\bar{x}_t)$ at the target, then the functions J_u are lower bounds on the cost-to-go: $J_u(x_u) \leq J_u^*(x_u)$ for all points $x_u \in \mathcal{X}_u$ and vertices $u \in \mathcal{V}$. To make the function J_s a tight lower bound on the cost-to-go J_s^* at the source point $\bar{x}_s$, we maximize the value $J_s(\bar{x}_s)$. We obtain the following program:

$$\max \quad J_s(\bar{x}_s) \tag{4a}$$

$$\text{s.t.} \quad J_v(x_v) \geq 0, \qquad \forall v \in \mathcal{V},\ x_v \in \mathcal{X}_v, \tag{4b}$$

$$J_u(x_u) \leq l_e(x_u, x_v) + l_v(x_v) + J_v(x_v), \quad \forall e = (u, v) \in \mathcal{E},\ (x_u, x_v) \in \mathcal{X}_e, \tag{4c}$$

$$J_t(\bar{x}_t) = l_t(\bar{x}_t), \tag{4d}$$

with variables J_v for $v \in \mathcal{V}$. The objective function (4a) maximizes the lower bound $J_s(\bar{x}_s)$, attaining the maximum when $J_s(\bar{x}_s) = J_s^*(\bar{x}_s)$. Thus the optimal solution of problem (4) yields the exact cost-to-go at the source point $\bar{x}_s$.

Simultaneously maximizing $J_v(x_v)$ across all points $x_v \in \mathcal{X}_v$ and vertices $v \in \mathcal{V}$ yields tight lower bounds on the cost-to-go over the entire GCS, akin to the classical many-to-one SPP. Furthermore, similar to the classical many-to-many SPP, program (4) can be generalized to produce lower-bounds on a cost-to-go $J^*_{s,t}(x_s, x_t)$ that is a function of the source and target vertices and points. This cost-to-go can be reused for multiple shortest-walk queries over the same GCS. For further details, see [43, App. A], which explores the many-to-many generalization of the SPP in GCS.

Program (4) is an infinite-dimensional Linear Program (LP), as both the constraints and the objective are linear in the functions J_v for $v \in \mathcal{V}$.

4.3 Efficient synthesis of cost-to-go lower bounds

We employ Semidefinite Programming (SDP) to produce an approximate solution of problem (4). We outline the approach and direct the reader to [43, §3.2] for more details.

The key challenge in program (4) lies in enforcing the inequality constraints (4b) and (4c): it is generally hard to require that a function is non-negative over a continuous set of points. However, if the function is polynomial and the set is basic semi-algebraic, then this constraint can be enforced in a convex (conservative) manner [5, §3.2.4]. Specifically, we restrict the lower bounds J_v of vertex v to be a convex quadratic function and search over its coefficients. Additionally, we restrict the convex sets $\mathcal{X}_v$ to be intersections of polyhedra and ellipsoids, and restrict the cost functions l_v and l_e to be convex quadratics (using quadratic approximations for non-quadratic functions). These restrictions allow us to cast program (4) as a tractable SDP, producing quadratic lower bounds on the cost-to-go function at each vertex. While arbitrary-degree polynomial lower bounds can be obtained via the sums-of-squares hierarchy [46, 47, 30], our computational experience is that quadratic polynomials strike a good balance between expressive power and computational complexity.

The cost-to-go synthesis is the most expensive part of our solution method. However, it is still tractable in practice, taking seconds to a minute even for complex systems and environments. Additionally, this expense becomes negligible in a multi-query setting, where the synthesis is performed only once and the cost-to-go is reused for multiple queries over the same GCS. This is particularly advantageous in applications where a robot must repeatedly solve similar planning problems within a static environment.

4.4 Extracting walks from the cost-to-go lower bounds

Similar to [43, 8], we extract an approximate solution to the SWP in GCS using incremental search and the cost-to-go lower bounds as a guiding heuristic. Specifically, we use multi-step lookahead greedy search, as it often produces feasible solutions quickly. We briefly describe this method below.

The walk is constructed incrementally, one vertex at the time. At iteration n of the search, let (v_n, x_n) be the current vertex and vertex point (initialized with $(s, \bar{x}_s)$), and

let k be the lookahead horizon. We consider all candidate k-step decision sequences (w', τ') that originate at (v_n, x_n) and, among them, greedily select the one that minimizes the k-step lookahead cost-to-go:

$$\min_{w',\,\tau'} \quad l(w', \tau') + J_{v_{n+k}}(x_{n+k}) \tag{5a}$$

$$\text{s.t.} \quad w' = (v_n, \dots v_{n+k}) \text{ is a } k\text{-step walk in } G, \tag{5b}$$

$$\tau' = (x_n, \dots x_{n+k}) \text{ is a trajectory along walk } w'. \tag{5c}$$

In (5b), we consider all k-step walks w' that originate at vertex v_n. Here we also include walks that reach the target vertex t in fewer than k steps. In (5c), each walk w' is associated with a trajectory τ' starting at the fixed point x_n and ending at a variable point x_{n+k} (or at the fixed point $\bar{x}_t$ if the walk w' reaches the target vertex t early). Together, (w', τ') is a candidate k-step decision sequence in the GCS. The objective (5a) minimizes the k-step lookahead cost-to-go: the cost of the k-step lookahead sequence $l(w', \tau')$ plus the remaining cost-to-go lower bound $J_{v_{n+k}}(x_{n+k})$.

To actually compute the solution (w', τ'), we solve multiple convex programs in parallel (one per candidate k-step walk w') and select the best. We take the first step of the selected walk, transitioning to (v_{n+1}, x_{n+1}), and proceed to next iteration. If program (5) is infeasible, we backtrack to a previous vertex and try alternative candidates. The process terminates upon reaching $(t, \bar{x}_t)$ or a fixed iteration budget. This method lacks guarantees of completeness or optimality, but we find it very effective in practice.

To improve the quality of these greedy solutions, we apply two post-processing steps. First, we reoptimize the trajectory along the candidate walk: it is convex program that yields a trajectory that is optimal within the walk. Second, we attempt to eliminate unnecessary cycles along the walk: if removing a cycle and reoptimizing the trajectory yields a solution with lower cost, we eliminate the cycle and continue.

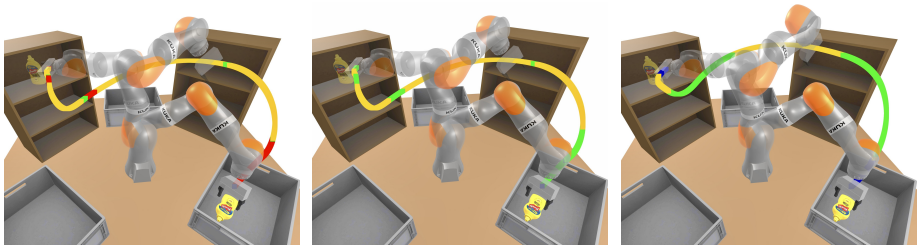
5 Numerical experiments

Mirroring the problem classes in Section 3, we illustrate three planning problems that are naturally formulated and effectively solved as a SWP in GCS. We also provide numerical comparisons between the shortest-walk and the shortest-path formulations, and defer qualitative comparisons with other algorithms to the next section.

All experiments are run on a laptop with an Apple M1 MAX chip with 16GB of RAM. We use Mosek10 [1] to solve all convex programs in this section and SNOPT7 [17] to solve non-convex programs in section Section 5.1, reporting parallelized solver time.

5.1 Robot-arm motion planning with acceleration limits

We consider a warehouse robot arm shown in Figure 6, tasked with moving items between shelves and bins. Given a pair of start and target conditions inside a shelf or a bin, the goal is to plan a minimum-time, smooth, collision-free trajectory that respects joint acceleration limits. Following Section 3.1, we cast this problem as both the SWP and the SPP in GCS, contrasting these formulations.



(a) SPP in GCS generates a velocity-limited trajectory of duration 2.21s, violating acceleration limits (red).

(b) Trajectory is post-processed to satisfy acceleration limits, increasing its duration to 2.57s.

(c) SWP in GCS enforces acceleration limits during search and yields a 2.25s acceleration-limited trajectory.

Fig. 6: Visual comparison of SPP in GCS with TOPP (a,b) and SWP in GCS (c) formulations. Both planners are tasked with finding an acceleration-limited trajectory from the top-left shelf to the right bin. The end-effector trajectory is red when the acceleration limits are violated, green when the acceleration is within 5% of the limit, yellow when the velocity is within 5% of the limit, and blue otherwise.

We first produce a convex decomposition of the collision-free configuration space using the IRIS-NP algorithm [48], taking advantage of IRIS clique seeding [67] to obtain regions that cover large volumes inside of the shelves and bins. This one-time step takes about 100 seconds, producing a connectivity graph with 16 vertices and 54 edges.

We follow the approach in [38] (see also Section 3.1) to cast the planning problem as an SPP in GCS and produce smooth trajectories through the collision-free regions. At each GCS vertex, we optimize the shape and time duration of a Bézier curve, as depicted in Figure 3a. The solution is a sequence of collision-free regions and a smooth, velocity-limited trajectory through them. Post-processing is required to incorporate acceleration limits. We consider two approaches: TOPP [62], which fixes the curves and reoptimizes their timing to satisfy the acceleration limits; and non-convex post-processing [63], which solves a non-convex program over the sequence of regions to enforce the limits.

For the SWP in GCS, we optimize a Bézier curve of fixed duration of 125ms at each vertex visit, enforcing acceleration limits in a convex manner during the incremental search. To compute the cost-to-go lower bounds, we employ the many-to-many generalization of (4) discussed in Sections 4.2 and 4.3, which takes 45s. The resulting lower bounds can be re-used across multiple queries over the same GCS.

We run 100 tests where the arm moves virtual items between shelves and bins, and compare solve times and time durations of the produced trajectories. The numerical results are reported in the first three rows of Table 1. The solve time of our SWP formulation takes on average 0.45s. On average, SPP with TOPP takes 30% longer, while SPP with non-convex post-processing takes 120% longer. As for the trajectory durations (solution cost), the differences are marginal, within 1-2% on average.

Figure 6 contrasts the SPP with TOPP post-processing and our SWP formulation. The former approach first generates a velocity-limited trajectory of 2.21s, shown in Figure 6a. This trajectory violates the acceleration limits, as shown by the red segments in

Problem, planner	Cost-to-go synthesis (s)	Solve time per query (s)	Solution cost	Success rate
V-A, SWP	45	0.45 (0.54)	2.62 (2.87)	100%
V-A, SPP w/ TOPP	N/A	0.66 (0.88)	2.64 (2.83)	100%
V-A, SPP w/ NC	N/A	1.00 (1.37)	2.60 (2.80)	100%
V-B, SWP	20.1	0.36 (2.67)	17.2 (23.5)	100%
V-B, SPP	N/A	Timed out	Timed out	0%
V-C, Fig.9a, SWP	2.1	0.18 (0.21)	20.9 (24.3)	100%
V-C, Fig.9a, SPP	N/A	2.83 (N/A)	21.7 (N/A)	61%
V-C, Fig.9b, SWP	1.5	0.49 (0.55)	32.5 (35.0)	100%
V-C, Fig.9b, SPP	N/A	Timed out	Timed out	0%

Table 1: Numerical results for the comparisons in Section 5. Where appropriate, we report the median over 100 trials, with the 90th percentile in the parenthesis. For each experiment, we use the time limit of 5 minutes.

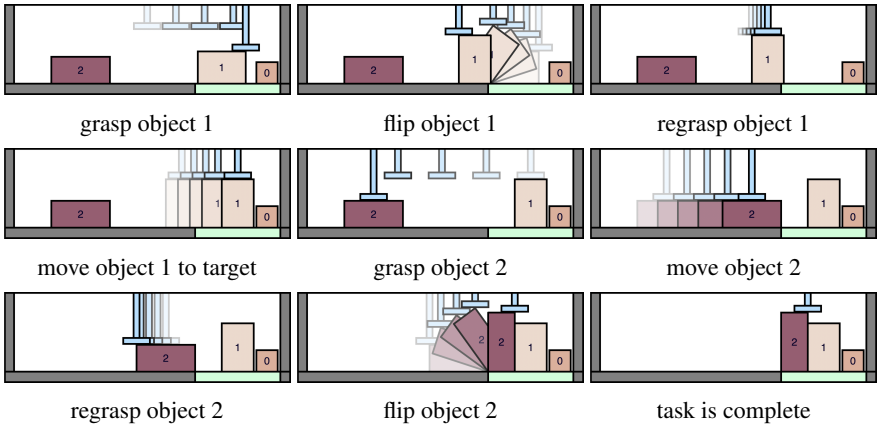


Fig. 7: Example of skill-chaining problem where a top-down suction-cup arm arranges three objects into the green target region on the right.

its end-effector trajectory. TOPP post-processing results in a slower trajectory (2.57s) that follows the same path but satisfies the acceleration limits (Figure 6b). Observe that the red segments of the end-effector trajectory become green, reflecting that the acceleration is within 5% of the limit. Our SWP formulation directly enforces acceleration limits during search: it produces a faster trajectory (2.25s) that saturates the acceleration limits for longer segments (green in Figure 6c). We also emphasize that, contrarily to TOPP, our method naturally handles constraints on higher-order derivatives, such as jerk or snap, which are important for industrial robot arms and drones.

Compared to the non-convex post-processing, our SWP formulation produces trajectories of comparable cost but much faster (and does not require a non-convex solver).

5.2 Skill chaining for top-down suction gripper

We now consider the object-arranging problem shown in Figure 7. The robot is a vertical suction-cup gripper, with position described by one horizontal coordinate. The environment contains three movable rectangular objects, described by width, height, and horizontal position. The robot can pick and place objects via top-center grasps, or flip them clockwise or counterclockwise by grasping their corners. The goal is to sort all objects into a target region (green horizontal intervals in Figure 7).

We cast this problem as an SWP in a GCS, following the skill chaining construction from Section 3.2. We define the configuration space to be the horizontal positions of the arm and the objects, along with the objects' dimensions. The robot's skills are: (1) moving the arm from one position to another while objects remain intact, (2) picking an object at its center and placing it at a collision-free location, (3) flipping an object clockwise or counterclockwise by grasping its corner, swapping its width and height.

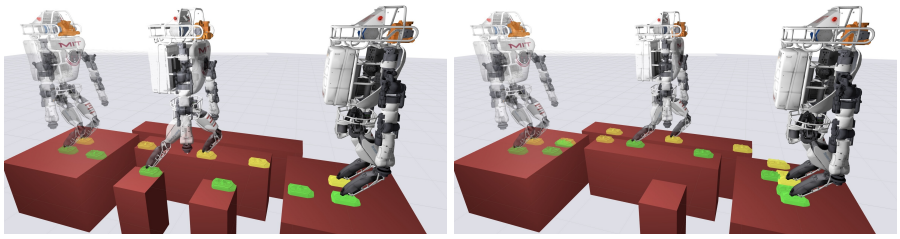
Each skill execution has cost equal to one plus the arm horizontal displacement. The sets $\mathcal{X}_\pi$ capture the feasible transitions under each skill. These sets are naturally non-convex due to the collision avoidance requirements and the combinatorial nature of selecting objects for manipulation. To address this, we decompose the skills into convex sub-skills, resulting in six arm movement, three pick-and-place, and six object-flipping skills. The GCS has a vertex for each sub-skill, a target vertex for any configuration with all objects in the target region, and six source vertices for any initial collision-free object arrangement. In total, we have 22 vertices and 120 edges. We employ the many-to-many generalization of (4), maximizing the average quadratic cost-to-go lower bounds over all source sets, which takes just 20 seconds. The resulting structure efficiently supports shortest-walk queries across various initial conditions and without fixing the order or precise placement of objects in the target region.

Figure 7 illustrates an example problem with its solution. Here, object 1 is already in the target region but must be moved and reoriented to also accommodate object 2. This is a challenging puzzle, as all three objects barely fit within the target region. The fourth and fifth rows of Table 1 report the solve-time statistics over a 100 randomized problems. For the SPP in GCS, the graph is constructed in a layered fashion as in Figure 5. Each vertex is duplicated over multiple layers, and each layer connects to the final target layer to allow transitions after any number of steps.

The SPP formulation requires duplicating vertices across layers because the number of steps is unknown a priori, resulting in a large mixed-discrete-continuous problem that is slow to generate and solve. In contrast, the incremental-search SWP formulation never constructs the full graph; it solves only small subproblems incrementally, while also taking advantage of parallelization. A crucial factor in this efficiency is the cost-to-go lower bound: without it, the search expands many more sub-walks and produces longer solutions. We emphasize again that these lower bounds are computed efficiently: the cost-to-go synthesis program captures skill repetition without vertex duplication and scales with the number of skills rather than the length of the solution sequence.

5.3 Hybrid footstep planning for a ZMP walker

Inspired by [10], we consider a footstep-planning problem for a humanoid robot navigating over stepping stones. The humanoid is shown in Figure 8 and must reach the



(a) A footstep plan across stepping stones. (b) Robot takes a detour due to missing stone.

Fig. 8: Footstep plans across stepping stones for the Atlas humanoid robot, produced in under 500ms. The SWP in GCS jointly optimizes for foot placement, contact forces, and centroidal dynamics, while maintaining stability enforced by the ZMP condition.

target by planning footstep locations and contact forces through the stepping stones, ensuring stability and avoiding foot collisions.

We use a Zero-Moment Point (ZMP) model for the robot dynamics [24, 64]. We constrain the ground contact forces to lie inside the friction cone and the robot ZMP to remain within the support polygon formed by the feet. This ensures that the robot can maintain static equilibrium at all times. We assume that the acceleration along the vertical axis is zero and that the robot does not rotate (zero angular acceleration). This results in an affine relationship between the ZMP and center-of-mass dynamics. The dynamics become piecewise affine when we account for footstep planning, with three primary contact modes: both feet on the ground, only the left foot, or only the right foot.

The footstep planning results in $O(N^2)$ potential contact modes (with N number of stones), but many are infeasible due to the constraints on the maximum distance between the feet. In practice, the number of modes grows as $O(DN)$, where D is the average number of adjacent stones. We assume that the feet do not rotate, although this could be easily modeled as in [10]. Additionally, we enforce constraints that prevent collisions between the feet. The resulting PWA system jointly considers safe footstep placement, contact forces, centroidal dynamics, and ZMP stability.

Following Section 3.3, we cast the optimal control problem for this PWA system as an SWP in GCS. We add a vertex for each PWA mode and a target vertex for the final state. The resulting GCS has 24 vertices and 58 edges for Figure 8a, and 21 vertices and 50 edges for Figure 8b. We compute quadratic cost-to-go lower bounds, maximizing their average across all vertices, which takes 2.1s and 1.5s, respectively. To cast the problem as the SPP in GCS, we again use the layered graph formulation from Figure 5.

We evaluate both formulations over 100 random problem instances, with initial conditions randomly chosen over the leftmost stone. The results are summarized in Table 1, rows 6 to 9. The SWP formulation takes on average 0.2s for the problem in Figure 8a and 0.5s for the one in Figure 8b, and has 100% success rate. The SPP formulation is significantly slower and even fails to solve some instances within the time limit of 5 minutes, primarily due to a very weak convex relaxation that leads to an extreme number of infeasible subproblems to consider. As before, vertex duplication makes the SPP in GCS expensive, whereas the incremental SWP formulation scales more effectively.

6 Discussion

In this paper, we have studied the SWP in GCS, proposed an efficient solution method for this problem, and shown how it captures a broad class of mixed discrete-continuous planning problems in robotics and control. The SWP in GCS complements the SPP in GCS, by better modeling problems involving discrete repetitions. While specialized algorithms may outperform the techniques proposed here in specific domains, the strength of the SWP in GCS lies in its broad applicability and consistent performance across diverse problem types. Below, we qualitatively compare it with related approaches.

For collision-free motion planning (Section 5.1), a quantitative comparison of the SPP in GCS and sampling-based planners, specifically PRM [26], is presented in [38]. This comparison shows that the SPP in GCS has a higher offline cost, but online it can produce better trajectories faster than PRM. As shown in Section 5.1, the SWP formulation proposed here exhibits comparable or improved computational performance to the SPP in GCS. Generally, sampling-based planners such as PRM, RRT [31], and more recent methods like VAMP [60], struggle to incorporate dynamic constraints or generate smooth trajectories. Trajectory optimization methods such as TrajOpt [52] and KOMO [61] address continuous dynamics but, without additional structure, lead to non-convex programs that converge slowly or can fail to find feasible solutions in cluttered environments. Recent GPU-based planners like cuRobo [57] improve runtime through massive parallelization, but inherit the same non-convexity and feasibility limitations, and typically do not optimize trajectory duration explicitly. In contrast, our SWP formulation preserves the advantages of the SPP in GCS (convex collision avoidance via Bézier parameterization and efficient reasoning over discrete mode sequences) while also capturing higher-order dynamic limits and smoothness in a convex manner.

For tightly constrained TAMP problems like in Section 5.2, integrated planning frameworks [15, 23, 28, 16, 21] often rely on custom samplers and carefully tuned distributions, which are expensive to construct. Feasible configurations and trajectories lie on lower-dimensional spaces, making naive random sampling ineffective. Optimization over such spaces is also challenging, thus approaches that rely on non-convex subroutines [61] are generally expensive. A main advantage of our shortest-walk formulation is that, if a convex decomposition is available, it relies on simple convex subroutines and requires no tuning of samplers or problem-specific heuristics. On the other hand, computing the convex decomposition can be challenging for some problems.

Finally, for optimal control of hybrid systems (Section 5.3), mixed-integer formulations [3, 42, 36, 33] remain the standard approach (beside the learning-based algorithms [65]). As shown in [35], many mixed-integer formulations are special cases of (or outperformed by) the SPP in GCS; our formulation builds on the same framework but naturally handles problems involving mode repetitions. Our method shares a common limitation with other mixed-integer approaches: combinatorial blow-up of contact modes. Contact-implicit methods [45, 32] effectively use randomized smoothing to mitigate this, but they remain local and cannot systematically reason over global discrete decisions. A hybrid strategy (i.e., smoothing some contact modes while treating others as explicit discrete transitions) could combine the strengths of both approaches.

Bibliography

[1] MOSEK ApS. *The MOSEK optimization toolbox for MATLAB manual. Version 10.1.*, 2024. URL http://docs.mosek.com/latest/toolbox/index.html.

[2] Federico Augugliaro, Angela P Schoellig, and Raffaello D'Andrea. Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach. In *2012 IEEE/RSJ international conference on Intelligent Robots and Systems*, pages 1917–1922. IEEE, 2012.

[3] Alberto Bemporad and Manfred Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3):407–427, 1999.

[4] John T Betts. Survey of numerical methods for trajectory optimization. *Journal of guidance, control, and dynamics*, 21(2):193–207, 1998.

[5] Grigoriy Blekherman, Pablo A Parrilo, and Rekha R Thomas. *Semidefinite optimization and convex algebraic geometry*. SIAM, 2012.

[6] Robert R Burridge, Alfred A Rizzi, and Daniel E Koditschek. Sequential composition of dynamically dexterous robot behaviors. *The International Journal of Robotics Research*, 18(6):534–555, 1999.

[7] Stephane Cambon, Rachid Alami, and Fabien Gravot. A hybrid approach to intricate motion, manipulation and task planning. *The International Journal of Robotics Research*, 28(1):104–126, 2009.

[8] Shao Yuan Chew Chia, Rebecca H Jiang, Bernhard Paus Graesdal, Leslie Pack Kaelbling, and Russ Tedrake. GCS*: Forward heuristic search on implicit graphs of convex sets. *arXiv preprint arXiv:2407.08848*, 2024.

[9] Thomas Cohn, Mark Petersen, Max Simchowitz, and Russ Tedrake. Non-Euclidean motion planning with graphs of geodesically-convex sets. *Robotics: Science and Systems*, 2023.

[10] Robin Deits and Russ Tedrake. Footstep planning on uneven terrain with mixed-integer convex optimization. In *2014 IEEE-RAS international conference on humanoid robots*, pages 279–286. IEEE, 2014.

[11] Robin Deits and Russ Tedrake. Computing large convex regions of obstacle-free space through semidefinite programming. In *Algorithmic Foundations of Robotics XI: Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics*, pages 109–124. Springer, 2015.

[12] Robin Deits and Russ Tedrake. Efficient mixed-integer planning for uavs in cluttered environments. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 42–49. IEEE, 2015.

[13] Mohamed Elbanhawi and Milan Simic. Sampling-based robot motion planning: A review. *IEEE Access*, 2:56–77, 2014.

[14] Enrique Fernandez-Gonzalez, Brian Williams, and Erez Karpas. Scottyactivity: Mixed discrete-continuous planning with convex optimization. *Journal of Artificial Intelligence Research*, 62:579–664, 2018.

[15] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. PDDLStream: Integrating symbolic planners and blackbox samplers via optimistic adap-

tive planning. In *Proceedings of the international conference on automated planning and scheduling*, volume 30, pages 440–448, 2020.

[16] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4(1):265–293, 2021.

[17] Philip E Gill, Walter Murray, and Michael A Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM review*, 47(1):99–131, 2005.

[18] Gustavo Goretkin, Alejandro Perez, Robert Platt, and George Konidaris. Optimal sampling-based planning for linear-quadratic kinodynamic systems. In *2013 IEEE International Conference on Robotics and Automation*, pages 2429–2436. IEEE, 2013.

[19] Bernhard P Graesdal, Shao YC Chia, Tobia Marcucci, Savva Morozov, Alexandre Amice, Pablo A Parrilo, and Russ Tedrake. Towards tight convex relaxations for contact-rich manipulation. *Robotics: Science and Systems*, 2024.

[20] Dylan Hadfield-Menell, Christopher Lin, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. Sequential quadratic programming for task plan optimization. In *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 5040–5047. IEEE, 2016.

[21] Kris Hauser and Jean-Claude Latombe. Multi-modal motion planning in non-expansive spaces. *The International Journal of Robotics Research*, 29(7):897–915, 2010.

[22] Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International journal of robotics research*, 34(7):883–921, 2015.

[23] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *2011 IEEE International Conference on Robotics and Automation*, pages 1470–1477. IEEE, 2011.

[24] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kiyoshi Fujiwara, Kensuke Harada, Kazuhito Yokoi, and Hirohisa Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In *2003 IEEE international conference on robotics and automation (Cat. No. 03CH37422)*, volume 2, pages 1620–1626. IEEE, 2003.

[25] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[26] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[27] George Konidaris and Andrew Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. *Advances in neural information processing systems*, 22, 2009.

[28] Athanasios Krontiris and Kostas E Bekris. Efficiently solving general rearrangement tasks: A fast extension primitive for an incremental sampling-based planner. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3924–3931. IEEE, 2016.

[29] Vince Kurtz and Hai Lin. Temporal logic motion planning with convex optimization via graphs of convex sets. *IEEE Transactions on Robotics*, 39(5):3791–3804, 2023.

[30] Jean B Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal on optimization*, 11(3):796–817, 2001.

[31] Steven LaValle. Rapidly-exploring random trees: A new tool for path planning. *Research Report 9811*, 1998.

[32] Simon Le Cleac'h, Taylor A Howell, Shuo Yang, Chi-Yen Lee, John Zhang, Arun Bishop, Mac Schwager, and Zachary Manchester. Fast contact-implicit model predictive control. *IEEE Transactions on Robotics*, 40:1617–1629, 2024.

[33] Jisun Lee, Hyungki Im, and Alper Atamtürk. Strong formulations for hybrid system control. *arXiv preprint arXiv:2412.11541*, 2024.

[34] Anirudha Majumdar and Russ Tedrake. Funnel libraries for real-time robust feedback motion planning. *The International Journal of Robotics Research*, 36(8): 947–982, 2017.

[35] Tobia Marcucci. *Graphs of Convex Sets with Applications to Optimal Control and Motion Planning*. PhD thesis, Massachusetts Institute of Technology, 2024.

[36] Tobia Marcucci and Russ Tedrake. Mixed-integer formulations for optimal control of piecewise-affine systems. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 230–239, 2019.

[37] Tobia Marcucci, Robin Deits, Marco Gabiccini, Antonio Bicchi, and Russ Tedrake. Approximate hybrid model predictive control for multi-contact push recovery in complex environments. In *2017 IEEE-RAS 17th international conference on humanoid robotics (Humanoids)*, pages 31–38. IEEE, 2017.

[38] Tobia Marcucci, Mark Petersen, David von Wrangel, and Russ Tedrake. Motion planning around obstacles with convex optimization. *Science robotics*, 8(84): eadf7843, 2023.

[39] Tobia Marcucci, Jack Umenberger, Pablo Parrilo, and Russ Tedrake. Shortest paths in graphs of convex sets. *SIAM Journal on Optimization*, 34(1):507–532, 2024.

[40] Tobia Marcucci, Mathew Halm, Will Yang, Dongchan Lee, and Andrew D Marchese. A biconvex method for minimum-time motion planning through sequences of convex sets. *arXiv preprint arXiv:2504.18978*, 2025.

[41] Daniel Mellinger, Alex Kushleyev, and Vijay Kumar. Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams. In *2012 IEEE international conference on robotics and automation*, pages 477–483. IEEE, 2012.

[42] Nicholas Moehle and Stephen Boyd. A perspective-based convex relaxation for switched-affine optimal control. *Systems & Control Letters*, 86:34–40, 2015.

[43] Savva Morozov, Tobia Marcucci, Alexandre Amice, Bernhard Paus Graesdal, Rohan Bosworth, Pablo A Parrilo, and Russ Tedrake. Multi-query shortest-path problem in graphs of convex sets. *arXiv preprint arXiv:2409.19543*, 2024.

[44] Ramkumar Natarajan, Chaoqi Liu, Howie Choset, and Maxim Likhachev. Implicit graph search for planning on graphs of convex sets. *arXiv preprint arXiv:2410.08909*, 2024.

[45] Tao Pang, HJ Terry Suh, Lujie Yang, and Russ Tedrake. Global planning for contact-rich manipulation via local smoothing of quasi-dynamic contact models. *IEEE Transactions on robotics*, 39(6):4691–4711, 2023.

[46] Pablo A Parrilo. *Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization*. California Institute of Technology, 2000.
[47] Pablo A Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Mathematical programming*, 96:293–320, 2003.
[48] Mark Petersen and Russ Tedrake. Growing convex collision-free regions in configuration space using nonlinear programming. *arXiv preprint arXiv:2303.14737*, 2023.
[49] Allen George Philip, Zhongqiang Ren, Sivakumar Rathinam, and Howie Choset. 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.
[50] Arthur Richards and Jonathan P How. Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In *Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301)*, volume 3, pages 1936–1941. IEEE, 2002.
[51] Tom Schouwenaars, Bart De Moor, Eric Feron, and Jonathan How. Mixed integer programming for multi-vehicle path planning. In *2001 European control conference (ECC)*, pages 2603–2608. IEEE, 2001.
[52] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270, 2014.
[53] Yasser Shoukry, Pierluigi Nuzzo, Alberto L Sangiovanni-Vincentelli, Sanjit A Seshia, George J Pappas, and Paulo Tabuada. SMC: Satisfiability modulo convex programming. *Proceedings of the IEEE*, 106(9):1655–1679, 2018.
[54] Eduardo Sontag. Nonlinear regulation: The piecewise linear approach. *IEEE Transactions on automatic control*, 26(2):346–358, 1981.
[55] Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 639–646. IEEE, 2014.
[56] Kaarthik Sundar and Sivakumar Rathinam. A* for graphs of convex sets. *arXiv preprint arXiv:2407.17413*, 2024.
[57] Balakumar Sundaralingam, Siva Kumar Sastry Hari, Adam Fishman, Caelan Garrett, Karl Van Wyk, Valts Blukis, Alexander Millane, Helen Oleynikova, Ankur Handa, Fabio Ramos, et al. curobo: Parallelized collision-free minimum-jerk robot motion generation. *arXiv preprint arXiv:2310.17274*, 2023.
[58] Richard S Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.
[59] Russ Tedrake et al. LQR-trees: Feedback motion planning on sparse randomized trees. In *Robotics: Science and Systems*, volume 2009, 2009.
[60] Wil Thomason, Zachary Kingston, and Lydia E Kavraki. Motions in microseconds via vectorized sampling-based planning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8749–8756. IEEE, 2024.
[61] Marc Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *IJCAI*, pages 1930–1936, 2015.

[62] Diederik Verscheure, Bram Demeulenaere, Jan Swevers, Joris De Schutter, and Moritz Diehl. Time-optimal path tracking for robots: A convex optimization approach. *IEEE Transactions on Automatic Control*, 54(10):2318–2327, 2009.

[63] David von Wrangel and Russ Tedrake. Using graphs of convex sets to guide nonconvex trajectory optimization. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9863–9870. IEEE, 2024.

[64] Miomir Vukobratović and Branislav Borovac. Zero-moment point—thirty five years of its life. *International journal of humanoid robotics*, 1(01):157–173, 2004.

[65] Jiankun Wang, Tianyi Zhang, Nachuan Ma, Zhaoting Li, Han Ma, Fei Meng, and Max Q-H Meng. A survey of learning-based robot motion planning. *IET Cyber-Systems and Robotics*, 3(4):302–314, 2021.

[66] Dustin J Webb and Jur Van Den Berg. Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics. In *2013 IEEE international conference on robotics and automation*, pages 5054–5061. IEEE, 2013.

[67] Peter Werner, Alexandre Amice, Tobia Marcucci, Daniela Rus, and Russ Tedrake. Approximating robot configuration spaces with few convex sets using clique covers of visibility graphs. *International Conference on Robotics and Automation*, 2024.

[68] Peter Werner, Thomas Cohn, Rebecca H Jiang, Tim Seyde, Max Simchowitz, Russ Tedrake, and Daniela Rus. Faster algorithms for growing collision-free convex polytopes in robot configuration space. *arXiv preprint arXiv:2410.12649*, 2024.

[69] Peter Werner, Richard Cheng, Tom Stewart, Russ Tedrake, and Daniela Rus. Superfast configuration-space convex set computation on gpus for online motion planning. *arXiv preprint arXiv:2504.10783*, 2025.

[70] Albert Wu, Sadra Sadraddini, and Russ Tedrake. R3T: Rapidly-exploring random reachable set tree for optimal kinodynamic planning of nonlinear hybrid systems. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4245–4251. IEEE, 2020.

[71] Lujie Yang, Tobia Marcucci, Pablo A Parrilo, and Russ Tedrake. A new semidefinite relaxation for linear and piecewise-affine optimal control with time scaling. *arXiv preprint arXiv:2504.13170*, 2025.

[72] Xiaojing Zhang, Alexander Liniger, and Francesco Borrelli. Optimization-based collision avoidance. *IEEE Transactions on Control Systems Technology*, 29(3): 972–983, 2020.