GCS*: Forward Heuristic Search on Implicit Graphs of Convex Sets

Shao Yuan Chew Chia, Rebecca H. Jiang, Bernhard Paus Graesdal,
Leslie Pack Kaelbling, and Russ Tedrake

Massachusetts Institute of Technology, Cambridge, MA 02139, USA
{shaoyuan, rhjiang, graesdal, lpk, russt}@mit.edu

Abstract. We consider large-scale, implicit-search-based solutions to Shortest Path Problems on Graphs of Convex Sets (GCS). We propose GCS*, a forward heuristic search algorithm that generalizes A* search to the GCS setting, where a continuous-valued decision is made at each graph vertex, and constraints across graph edges couple these decisions, influencing costs and feasibility. Such mixed discrete-continuous planning is needed in many domains, including motion planning around obstacles and planning through contact. This setting provides a unique challenge for best-first search algorithms: the cost and feasibility of a path depend on continuous-valued points chosen along the entire path. We show that by pruning paths that are cost-dominated over their entire terminal vertex, GCS* can search efficiently while still guaranteeing cost-optimality and completeness. To find satisficing solutions quickly, we also present a complete but suboptimal variation, pruning instead reachability-dominated paths. We implement these checks using polyhedral-containment or sampling-based methods. The former implementation is complete and cost-optimal, while the latter is probabilistically complete and asymptotically cost-optimal and performs effectively even with minimal samples in practice. We demonstrate GCS* on planar pushing tasks where the combinatorial explosion of contact modes renders prior methods intractable and show it performs favorably compared to the state-of-the-art. Project website: shaoyuan.cc/research/gcs-star/

Keywords: Graph Search · Task and Motion Planning · Manipulation · Convex Optimization · Algorithmic Completeness and Complexity.

1 Introduction

Many real-world planning problems involve making discrete and continuous decisions jointly. Collision-free motion planning selects whether to go left or right around an obstacle along with a continuous trajectory to do so. In Task and Motion Planning (TAMP), discrete task-level decisions about the type and sequence of actions are intimately coupled with continuous robot motions and object configurations. For example, where a robot grasps a hockey stick impacts its ability to hook an object [13]; in the construction of a tower, the order in which materials are assembled, as well as their geometric relationships, affect stability [8,25].

A natural representation of such a problem structure is the Graph of Convex Sets (GCS) [15]. In a GCS, graph vertices represent these discrete choices, with
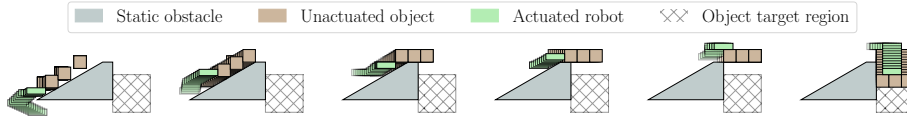
Fig. 1: An ϵ-suboptimal solution found in 21.9s by GCS* using sampling-based REACHESCHEAPER domination checks on the STACK planar pushing task. STACK is formulated as a GCS problem with approximately 1.3×10^9 vertices and up to 8.5×10^{17} edges.

edges denoting allowed transitions. A continuous decision within each vertex is encoded by a *point* constrained to lie within a convex set associated with the vertex. For example, the free configuration space around an obstacle may be (sometimes approximately) decomposed into a number of intersecting convex regions in which trajectory segments can be planned. The point associated with each vertex, which encodes the motion within the corresponding free-space region, could be, for example, a sequence of positions at a fixed number of knot points [14]. In the case of a single knot point, the associated vertex's convex set is simply the corresponding convex free-space set. In the case of more knot points, the vertex's set would be the higher-dimensional set in which each knot point is contained within the free-space set. In this example, intersecting free-space regions lead to graph edges. A valid way to get around the obstacle involves a *path* – a sequence of vertices – through the graph, along with a *trajectory* – a sequence of continuous-valued points assigned to vertices on the path. Importantly, in such problems, there may be constraints coupling points across edges, presenting a fundamental difference from classical graph search problems. For example, for trajectory continuity, knot points across an edge must coincide.

While Shortest-Path Problems (SPPs) in discrete graphs can be solved in polynomial time, SPPs in GCS are NP-hard [15]. GCS can be solved with a Mixed-Integer Convex Program (MICP) formulation; Marcucci et al. give a notably scalable transcription of SPPs in GCS with a tight convex relaxation [15]. However, these approaches become intractable as the graph grows. Many discrete-continuous problems exhibit a combinatorial explosion of discrete modes, e.g., in a manipulation task where contact dynamics between every pair of bodies are considered. In TAMP, the search space is typically so large that explicit enumeration of the search space is impractical. For such problems, we can instead define a graph implicitly via a source vertex and a successor operator that acts on a vertex and returns all outgoing edges. Implicit graph search algorithms use this representation to solve SPPs without loading the entire graph in memory. A* [7] is forward heuristic search algorithm which can solve the SPP on implicit discrete graphs. However, key differences between discrete graphs and GCS make search in GCS more challenging.

The SPP on a discrete graph has the optimal substructure property [2]: All sub-paths of an optimal path between two vertices are themselves optimal paths between the first and last vertices they traverse. A* (or any best-first search algorithm) leverages this property, pruning a sub-path when its cost-to-come is dominated by another sub-path terminating in the same vertex because it cannot be a sub-path of any optimal path [22]. However, the SPP on a GCS lacks
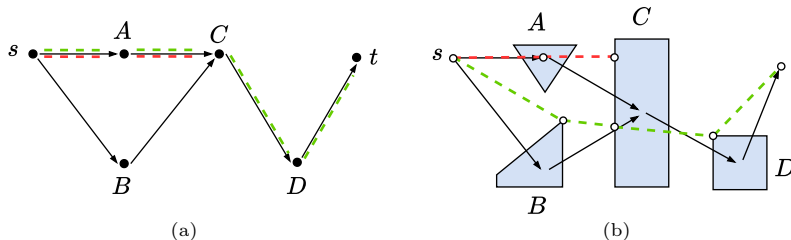
Fig. 2: An abstract discrete graph (a) and GCS (b), for example, mapping to the path planning problem for a hopping robot tasked to hop from s to t across polygonal stepping stones. Arrows are edges between vertices. Vertices are represented as black dots in (a) and via their (blue) convex sets in (b). White dots are trajectory points on the GCS. Edge costs are the Euclidean distances traversed between vertices. A discrete graph (a) has the optimal substructure property while a GCS (b) does not. In (a), the optimal path between s and C (red) is a sub-path of the optimal path between s and t (green), which goes through C. In (b), the optimal path between s and C is not a sub-path of the optimal path between s and t, which goes through C. However, each *point* on the green path's trajectory is reached optimally on the green path.

the optimal substructure property. Importantly, as exemplified in fig. 2 and by Morozov et al. [16], *a sub-path of an optimal path on a GCS is not necessarily an optimal path between that subpath's first and last vertices.* Thus, when considering whether a partial path to the target could be a sub-path of an optimal path, it is insufficient to ask whether it is the cheapest way to reach its terminal vertex. As a result, a naive application of A* to GCS could prune an important sub-path, preventing the optimal path from being returned. We leverage the key insight that, while the optimal substructure property does not hold on the path level in GCS, it does hold on the trajectory level: Given an optimal path and corresponding trajectory, any sub-path is optimal to reach its own final point, as exemplified in fig. 2 (b). That is, while an optimal path may not contain an optimal subpath to reach each *vertex* it visits, its trajectory is optimal to reach each *point* it visits. In order to maintain this property, we place no restrictions on the number of times a path may revisit vertices [16]. Formally, paths that may revisit vertices are generally referred to as "walks," but for readability we continue to use "path" even though this term traditionally excludes cycles.

The trajectory-level optimal substructure property holds because the optimal way to reach a point is independent of the optimal way to continue on from it. However, this is only true if visiting a particular vertex on the way to that point does not exclude that vertex from being visited subsequently. It is well-known that shortest paths on discrete graphs never revisit vertices [2]; however, this does not hold on a GCS because a different point may be visited each time.

More severe than the loss of optimality is the loss of completeness. Because a GCS may have constraints coupling points across edges, a path may not be feasible even if edges connect the sequence of vertices. As a result, A* on a GCS might prune a candidate path even though the cheaper alternate path is infeasible for reaching the target. An example is shown in fig. 3.

In this paper, we address these challenges and generalize A* to the GCS setting. We call our algorithm GCS* (sec. 4), leveraging two key insights. First, to retain optimality, the algorithm must keep track of every path that reaches *any*
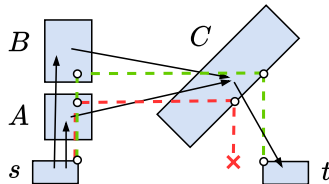
Fig. 3: A contrived example of a GCS where points spanning edges (s, A), (s, B), and (C, t) are constrained to be vertically aligned, and points spanning (A, C) and (B, C) horizontally aligned. For example, representing a path planning problem for a hopping robot that can only hop in axis-aligned directions. A* prunes $[s, B, C]$ since it has higher cost-to-come than $[s, A, C]$. However, $[s, A, C, t]$ is infeasible (red), while $[s, B, C, t]$ is feasible (green).

point in a set more cheaply than alternate paths. Second, to retain completeness, the algorithm only needs to keep track of every path that reaches previously unreached points within a set. These insights give rise to domination checks we call REACHESCHEAPER and REACHESNEW, respectively (sec. 4.2). Using REACHESCHEAPER with an admissible heuristic makes GCS* optimal, while using REACHESNEW prunes paths more aggressively, sacrificing optimality for speed, but retaining completeness[1]. We present polyhedral-containment-based implementations of these checks and prove completeness and optimality of GCS* (sec. 4.3). However, we achieve significant speed improvements with only mildly suboptimal results with sampling-based implementations, under which GCS* is probabilisitically complete and asymptotically cost optimal, i.e., complete and optimal in the limit of infinite samples. The theoretical properties of each variant of the algorithm are summarized in table 2.

Finally, we formulate planar pushing as a GCS problem in sec. 5 and share GCS* results for tasks with multiple moving objects and multi-step solutions in sec. 6. Such problems are notorious for combinatorially exploding discrete options. For example, the STACK task (fig. 1) leads to approximately 10^9 sets and up to 10^{18} edges, making it wildly intractable for methods that require loading the graph into memory, but sampling-based GCS* finds a solution in 21.9 seconds. Two key strengths of the best-first search framework extend to GCS*. First, GCS* is able to solve huge problems that are intractable for methods which require explicit construction of the graph. Second, search effort is proportional to query difficulty. In contrast, methods that solve the problem as a single optimization program require full effort regardless of query difficulty. Similar to A*, GCS* also guides the search using a priority function that sums the cost-to-come and heuristic cost-to-go. Exactly as in weighted A*, inflation of the heuristic can be used to trade-off bounded sub-optimality with speed [21].

2   Related Work

We overview three approaches to discrete-continuous planning: sampling-based, optimization-based, and hybrid methods that combine graph search with sampling or optimization. Sampling-based methods, like Rapidly-exploring Random Trees [12] and Probabilistic Roadmap Method [11] were developed for path

---

[1] Similar to A* [22], GCS* is complete if a solution exists or the user defines a limit on the length of paths that can be returned.

planning with obstacles and extended for contact-rich manipulation [1,19]. Unlike nonconvex trajectory optimization, which can fail to find feasible solutions, sampling-based methods are typically probabilistically complete. However, incorporating continuous differential constraints on discrete samples is challenging, often making kinodynamic versions of these algorithms less effective [14].

To achieve both the completeness of sampling-based methods and trajectory optimization's ease of handling continuous constraints, some have proposed posing the entire problem as a single MICP, solving a convex relaxation, and applying a cheap rounding step to recover a feasible solution. This strategy has been applied to motion planning around obstacles for high-dimensional robotic systems [14] and planar pushing of a single object [6]. We extend these benefits to the class of problems too large be posed as a single optimization program.

To solve large problems modelled implicitly, many TAMP algorithms perform a tree search, alternating between choosing a discrete action and sampling continuous action parameters [5]. However, using sampling to determine feasibility of a sequence of abstract actions does not provide a guaranteed answer in finite time. Other hybrid methods combine graph search with optimization over continuous variables. Using optimization to set path expansion priority allows continuous considerations to closely inform discrete search. Logic-Geometric Programming [25,26] uses non-linear programming (NLP) to guide the search, but the non-convexity of NLP leads to the loss of completeness and optimality. We build on these ideas, additionally requiring a convex decomposition of configuration space. While this can be non-trivial to generate [20,3,27], convex optimization allows us to determine the feasibility of a path in GCS absolutely, and perform domination checks that enable completeness and global optimality.

Recent works have proposed solving SPPs on GCS with graph search. A^*-GCS [24] solves the convex relaxation on increasingly large sub-graphs, using successive solutions to guide sub-graph growth. In this work, we instead iteratively find solutions for sub-paths rather than sub-graphs, which scales better to larger graphs. INSATxGCS (IxG) and IxG* [18] adapt INterleaved Search and Trajectory Optimization (INSAT) [17] for collision-free GCS motion planning [14]. These algorithms demonstrate impressive speedups over the convex relaxation and rounding approach [14] across a range of tasks including multi-arm assembly. However, IxG is only complete subject to the assumption that all edge constraints along any path in the graph can be satisfied; In reality, individually feasible edge constraints can still be jointly infeasible, as in fig. 3. Edge constraints add significant expressivity to GCS and are heavily used, including by Marcucci et al. [14] and Graesdal et al. [6]. While IxG* is complete without this assumption, it uses IxG to generate an upper bound on the optimal cost. Supposing IxG fails to find a feasible path, IxG* will use a trivial (infinite) upper bound. Then IxG* will not prune *any* paths, rendering large problems intractable. Furthermore, while the main search of IxG and IxG* can be done on implicit graphs, both methods use a precomputed heuristic that requires explicit graphs. We build on the this approach, addressing the case where graphs are too large to be explicitly constructed, e.g. the STACK task (fig. 1).

3   Problem Formulation

In this section we present the problem of interest as a mathematical program.

Table 1: Summary of mathematical notation

$G := (\mathcal{V}, \mathcal{E})$	Directed graph with vertices $\mathcal{V}$ and edges $\mathcal{E}$
$\mathcal{X}_v$	Convex set associated with vertex v
$\mathcal{X}_e$	Convex constraint set for edge e
s, t	Source and target vertices
$\mathbf{v}$	Discrete path in the graph, defined as a sequence of vertices
$\mathbf{v}_i$	The i-th vertex in the path $\mathbf{v}$
$\mathbf{v}_{\text{end}}$	Last vertex in the path $\mathbf{v}$
$\mathcal{E}_{\mathbf{v}}$	Set of edges traversed by path $\mathbf{v}$
$\mathbf{x}$	Continuous trajectory
$\mathbf{x}_i$	The i-th point on trajectory $\mathbf{x}$
$\mathbf{x}_i \in \mathcal{X}_v$	The point on trajectory $\mathbf{x}$ associated with vertex v
$c(\mathbf{x}_u, \mathbf{x}_v)$	Cost of the edge between vertices u and v, evaluated on trajectory $\mathbf{x}$
$f^*(v)$	Optimal cost of a solution via vertex v
$g^*(x)$	Optimal cost-to-come from s to point x
$h^*(x)$	Optimal cost-to-go from point x to t
$\tilde{f}(\mathbf{v})$	Total cost estimate via path $\mathbf{v}$
$\tilde{f}(\mathbf{v}, x)$	Total cost estimate via path $\mathbf{v}$ and point $x \in \mathcal{X}_{\mathbf{v}_{\text{end}}}$
$\tilde{g}(\mathbf{v}, x)$	Optimal cost-to-come from s to point $x \in \mathcal{X}_{\mathbf{v}_{\text{end}}}$ via path $\mathbf{v}$
$\tilde{h}(x)$	Heuristic estimate of cost-to-go from point x to t
$\mathbf{x}^*(\mathbf{v})$	Optimal trajectory via path $\mathbf{v}$ subject to the heuristic $\tilde{h}$

A GCS [15] is a directed graph $G := (\mathcal{V}, \mathcal{E})$ defined by a (potentially infinite) set of vertices $\mathcal{V}$ and edges $\mathcal{E} \subset \mathcal{V}^2$, where $(u, v) \in \mathcal{E}$ if the graph allows transitions from vertex u to vertex v, i.e., v is a *successor* of u. Each vertex $v \in \mathcal{V}$ is paired with a compact convex set $\mathcal{X}_v$. These sets may live in different spaces. We assume each vertex has a finite number of successors. The graph is implicitly defined via source vertex $s \in \mathcal{V}$ and operator SUCCESSORS that, when applied to a vertex u, returns all successors v_i of u. We define a path $\mathbf{v}$ in graph G as a sequence of vertices, where the bold font indicates a sequence. A subscripted index operates on a sequence to select the element at that index: $\mathbf{v}_i$ gives the i-th vertex in the sequence and each $\mathbf{v}_{i+1}$ is a successor of $\mathbf{v}_i$. Likewise, we define a trajectory $\mathbf{x}$ on path $\mathbf{v}$ as a sequence of points in the sets associated with $\mathbf{v}$, $\mathbf{x}_i \in \mathcal{X}_{\mathbf{v}_i}$. We also overload the indexing of $\mathbf{x}$ to go by vertex, such that $\mathbf{x}_{\mathbf{v}_i} := \mathbf{x}_i$. We denote the last vertex in the path $\mathbf{v}$ as $\mathbf{v}_{\text{end}}$, and the edges traversed by $\mathbf{v}$ as $\mathcal{E}_{\mathbf{v}}$. The cost of edge $e := (u, v)$ is determined by a proper, closed, convex, positive, bounded-away-from-zero function of its endpoints $c(\mathbf{x}_u, \mathbf{x}_v)$. The bounded-away-from-zero stipulation ensures finite optimal paths when cycles are permitted. Each edge can additionally have constraints $(\mathbf{x}_u, \mathbf{x}_v) \in \mathcal{X}_e$, where $\mathcal{X}_e$ is a closed convex set. For source vertex $s \in \mathcal{V}$ and target vertex $t \in \mathcal{V}$, the SPP in GCS is

$$\underset{\mathbf{v}, \, \mathbf{x}}{\text{minimize}} \quad \sum_{(u,v) \in \mathcal{E}_{\mathbf{v}}} c(\mathbf{x}_u, \mathbf{x}_v) \tag{1a}$$

$$\text{subject to} \quad \mathbf{v}_0 = s, \; \mathbf{v}_{\text{end}} = t, \tag{1b}$$

$$\mathbf{v}_{i+1} \in \text{SUCCESSORS}(\mathbf{v}_i), \qquad \forall i \in [0, \text{end} - 1], \tag{1c}$$

$$\mathbf{x}_v \in \mathcal{X}_v, \qquad \forall v \in \mathbf{v}, \tag{1d}$$

$$(\mathbf{x}_u, \mathbf{x}_v) \in \mathcal{X}_e, \qquad \forall e := (u, v) \in \mathcal{E}_{\mathbf{v}}. \tag{1e}$$

The objective (1a) is to minimize the total cost of traversing the path $\mathbf{v}$ in the graph. Constraint (1b) enforces that path $\mathbf{v}$ starts at source vertex s, and ends at target vertex t. Constraint (1c) enforces that each edge in the path $\mathbf{v}$ exists. Constraint (1d) enforces that each point on the trajectory $\mathbf{x}$ lies within the convex set corresponding to its vertex. Finally, constraint (1e) enforces that the continuous values of trajectory $\mathbf{x}$ satisfy all edge constraints along path $\mathbf{v}$.

If the discrete path $\mathbf{v}$ is fixed, prog. (1) becomes convex and easy to solve; we call this program CONVEXRESTRICTION. Given $\mathbf{v}$, solving CONVEXRESTRICTION determines its corresponding optimal cost and optimal trajectory $\mathbf{x}$.

4 Approach

4.1 Graph Search Formulation

A* (alg. 1) searches over paths from the source, pruning some and expanding others based on estimated costs of extending to reach the target, until a path reaches the target and is returned. In particular, paths await expansion in a priority queue Q ordered by total cost estimate $\tilde{f}$ (line 6). Often, instead of specifying $\tilde{f}$ directly, a user equivalently provides $\tilde{h}$, a heuristic that estimates costs-to-go from vertices to the target, and $\tilde{f}$ is computed as the sum of heuristic $\tilde{h}$ and cost-to-come $\tilde{g}$. A path is pruned via a domination check if its cost-to-come $\tilde{g}$ is greater than that of the current cheapest path reaching the same vertex (line 10). This current cheapest path for each vertex is stored in a map S (line 11), that maps each vertex to the current lowest-cost path. Equivalently, $\tilde{f}$ can be compared since paths reaching the same vertex share an $\tilde{h}$ value. We use the term "domination check" generically to refer to a function that determines whether a candidate path should be pruned or added to Q. We call a path *expanded* if it has been popped from Q (line 6). We call a vertex *expanded* if any path terminating at it has been expanded.

GCS* (alg. 2) proceeds similarly. The key difference is the use of a different domination check NOTDOMINATED $\in$ {REACHESCHEAPER, REACHESNEW} (defined in sec. 4.2). To facilitate these domination checks, GCS* maintains a map S which maps each vertex to a set of un-pruned paths reaching the vertex. This is in contrast to A*, which stores a single path reaching each vertex. For any vertex $v \in \mathcal{V}$, we define $f^*(v)$ as the optimal cost of prog. (1) with the additional constraint that vertex v is on the path $\mathbf{v}$ (not necessarily as the terminal vertex $\mathbf{v}_{\text{end}}$), i.e., $v \in \mathbf{v}$, where path $\mathbf{v}$ is a decision variable in prog. (1). Let $g^*(x)$ be the optimal cost-to-come from source s to point x , and $h^*(x)$ be the optimal cost-to-go from point x to target t, both infinite if infeasible.[2] If point $x \in \mathcal{X}_v$ is on the optimal path through vertex v, we have $f^*(v) = g^*(x) + h^*(x)$. Note that $f^*(x)$, $g^*(x)$, and $h^*(x)$ are well defined even if point x is contained in (intersecting) sets corresponding to multiple vertices.

---

[2] More precisely, every "function" $l(x)$ that takes a point x as input in this paper is actually a family of functions $\{l_v : \mathcal{X}_v \to \mathbb{R} \mid v \in V\}$ defined over each vertex, as these points may lie in different spaces. These functions are defined to agree across vertices when these vertex sets share points, $l_{v_1}(x) = l_{v_2}(x) \ \forall x \in \mathcal{X}_{v_1} \cap \mathcal{X}_{v_2}$. For clarity, we drop the subscript and treat $l(x)$ as a single function.

In general, f^*, g^*, and h^* cannot be computed without solving prog. (1). Instead, we define computationally viable functions $\tilde{f}$, $\tilde{g}$ and $\tilde{h}$. The A* algorithm uses a heuristic function for estimating the cost-to-go from a vertex to the target. Extending this to GCS*, we use a heuristic function $\tilde{h}(x)$ that estimates the cost-to-go from a point x to the target t.

Next, we define $\tilde{g}$ and $\tilde{f}$. No longer referring to the decision variables from prog. (1), let $\mathbf{v}$ be a candidate partial path to the target t such that $\mathbf{v}_0 = s$, but $\mathbf{v}_{\text{end}}$ is not necessarily t. Let $\mathbf{x}^*(\mathbf{v})$ denote an optimal trajectory through $\mathbf{v}$ subject to the heuristic $\tilde{h}$ assigning the cost from the final point in the optimal partial trajectory $\mathbf{x}^*(\mathbf{v})_{\text{end}}$ to target t. Then $\tilde{g}(\mathbf{v}, x)$ for $x \in \mathcal{X}_{\mathbf{v}_{\text{end}}}$ is the optimal cost-to-come from source s to point x via partial path $\mathbf{v}$. Then, the total cost estimate via a partial path $\mathbf{v}$ is defined to be $\tilde{f}(\mathbf{v}) := \tilde{g}(\mathbf{v}, \mathbf{x}^*(\mathbf{v})_{\text{end}}) + \tilde{h}(\mathbf{x}^*(\mathbf{v})_{\text{end}})$.

Algorithm 1 Discrete A*	**Algorithm 2** GCS*
Input: s, t, $\tilde{f}$, $\tilde{g}$, SUCCESSORS	**Input:** s, t, $\tilde{f}$, NOTDOMINATED, SUCCESSORS
Output: Path from s to t or Fail	**Output:** Path from s to t or Fail
1: $\mathbf{v} = [s]$	1: $\mathbf{v} = [s]$
2: $S \leftarrow \{s : \mathbf{v}\}$ ▷ *Map of paths*	2: $S \leftarrow \{s : \{\mathbf{v}\}\}$ ▷ *Map of sets of paths*
3: $Q \leftarrow$ priority queue ordered by $\tilde{f}(\mathbf{v})$	3: $Q \leftarrow$ priority queue ordered by $\tilde{f}(\mathbf{v})$
4: $Q.\text{ADD}(\mathbf{v})$	4: $Q.\text{ADD}(\mathbf{v})$
5: **while** $Q \neq \emptyset$ **do**	5: **while** $Q \neq \emptyset$ **do**
6: $\mathbf{v} \leftarrow Q.\text{POP}()$	6: $\mathbf{v} \leftarrow Q.\text{POP}()$
7: **if** $\mathbf{v}_{\text{end}} = \text{t}$ **then return** $\mathbf{v}$	7: **if** $\mathbf{v}_{\text{end}} = \text{t}$ **then return** $\mathbf{v}$
8: **for all** $v' \in \text{SUCCESSORS}(\mathbf{v}_{\text{end}})$ **do**	8: **for all** $v' \in \text{SUCCESSORS}(\mathbf{v}_{\text{end}})$ **do**
9: $\mathbf{v}' = [\mathbf{v}, v']$	9: $\mathbf{v}' = [\mathbf{v}, v']$
10: **if** $v' \notin S$ or $\tilde{g}(\mathbf{v}') < \tilde{g}(S[v'])$ **then**	10: **if** NOTDOMINATED$(\mathbf{v}', S[v'])$ **then**
11: $S[v'] = \mathbf{v}'$	11: $S[v'].\text{ADD}(\mathbf{v}')$
12: $Q.\text{ADD}(\mathbf{v}')$	12: $Q.\text{ADD}(\mathbf{v}')$
13: **return** Fail	13: **return** Fail

To evaluate the total cost estimate for a path additionally restricted to pass through a specific point $x \in \mathcal{X}_{\mathbf{v}_{\text{end}}}$, we overload $\tilde{f}(\mathbf{v}, x) := \tilde{g}(\mathbf{v}, x) + \tilde{h}(x)$. CONVEXRESTRICTION can be used to evaluate $\tilde{g}(\mathbf{v}, x)$ and $\tilde{f}(\mathbf{v}, x)$ (used to evaluate sample points in sec. 4.2). Additionally, if $\tilde{h}$ is convex[3], CONVEXRESTRICTION can be used to evaluate $\mathbf{x}^*(\mathbf{v})$ (used to return the final optimal trajectory) and $\tilde{f}(\mathbf{v})$ (used to prioritize the queue Q in alg. 2).

We assume heuristic $\tilde{h}(x)$ is nonnegative, i.e., $\tilde{h}(x) \geq 0$, $\forall x$. Furthermore, heuristic $\tilde{h}$ must be *pointwise admissible* (def. 1) in order for optimality guarantees (sec. 4.3) to hold. We extend the classical definition of admissibility over vertices to a pointwise definition:

---

[3] More precisely, $\tilde{h}_v$ is convex for all $v \in \mathcal{V}$.

Definition 1. *A heuristic function $\tilde{h}$ is pointwise admissible if*

$$\tilde{h}(x) \leq h^*(x), \quad \forall x \in \mathcal{X}_v, \quad \forall v \in \mathcal{V}.$$

4.2   Domination Checks

In line 10 of alg. 2, GCS* uses one of two domination checks, ReachesCheaper or ReachesNew. In practice, exact checks are not tractable, so we compute approximate checks. An approximate NotDominated check is said to be *conservative* if it never returns False incorrectly, i.e., it never says a candidate path is dominated when it is not. Under a conservative domination check, GCS* may track more candidate paths than necessary, but never overlooks an important path (which paths are "important" depends on the domination check being used and will be precisely stated in this section). Using exact or conservative ReachesCheaper checks, GCS* is cost optimal. Satisficing solutions (solutions which are feasible but likely suboptimal) can be found more quickly using ReachesNew. Using exact or conservative ReachesNew checks, GCS* is complete.
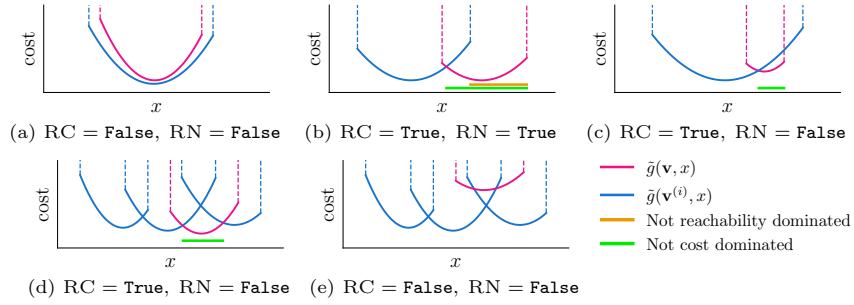

(a) RC = False, RN = False   (b) RC = True, RN = True   (c) RC = True, RN = False

(d) RC = True, RN = False   (e) RC = False, RN = False

—— $\tilde{g}(\mathbf{v}, x)$
—— $\tilde{g}(\mathbf{v}^{(i)}, x)$
—— Not reachability dominated
—— Not cost dominated

Fig. 4: Scenarios showing whether a candidate path $\mathbf{v}$ (pink) ReachesCheaper (RC) and ReachesNew (RN) compared to alternate paths $\{\mathbf{v}^{(i)} | i \in \{1, 2, 3\}\}$ (blue), where the dimension of the set $\mathcal{X}_{\mathbf{v}_{\mathrm{end}}}$ is 1. Green and orange lines denote points x such that eq. (2) and eq. (3) hold, respectively. Points x for which the cost-to-come $\tilde{g}(\mathbf{v}, x)$ is not drawn (outside dashed lines) are unreachable via path $\mathbf{v}$ due to constraints, and $\tilde{g}(\mathbf{v}, x) = \infty$.

If a candidate path $\mathbf{v}$ reaches some point $x \in \mathcal{X}_{\mathbf{v}_{\mathrm{end}}}$ cheaper than any way found yet, as in fig. 4 (b), (c), and (d), we say it ReachesCheaper:

$$\begin{aligned} \textsc{ReachesCheaper}(\mathbf{v}, S[\mathbf{v}_{\mathrm{end}}]) \;\; := \;\; &\exists\, x \in \mathcal{X}_{\mathbf{v}_{\mathrm{end}}} \\ &\text{s.t. } \tilde{g}(\mathbf{v}, x) < \tilde{g}(\mathbf{v}', x),\, \forall\, \mathbf{v}' \in S[\mathbf{v}_{\mathrm{end}}]. \end{aligned} \quad (2)$$

If ReachesCheaper$(\mathbf{v}, S[\mathbf{v}_{\mathrm{end}}])$ = False, path $\mathbf{v}$ is said to be *dominated*, or, more specifically, *cost-dominated*. Intuitively, GCS* returns an optimal path when it has pruned only cost-dominated paths (discussed in sec. 4.3, proven in appendix A) because such paths cannot be subpaths of optimal paths.

If a candidate path $\mathbf{v}$ reaches some point $x \in \mathcal{X}_{\mathbf{v}_{\mathrm{end}}}$ that has not yet been reached, as in fig. 4 (b), we say it ReachesNew:

$$\begin{aligned} \textsc{ReachesNew}(\mathbf{v}, S[\mathbf{v}_{\mathrm{end}}]) \;\; := \;\; &\exists\, x \in \mathcal{X}_{\mathbf{v}_{\mathrm{end}}} \\ &\text{s.t. } (\tilde{g}(\mathbf{v}, x) < \infty) \,\wedge\, (\tilde{g}(\mathbf{v}', x) = \infty,\, \forall\, \mathbf{v}' \in S[\mathbf{v}_{\mathrm{end}}]). \end{aligned} \quad (3)$$

If $\text{REACHESNEW}(\mathbf{v}, S[\mathbf{v}_{\text{end}}]) = \texttt{False}$, $\mathbf{v}$ is said to be *dominated*, or, more specifically, *reachability-dominated*. Where REACHESCHEAPER compares costs-to-come at all points in the terminal set, REACHESNEW compares feasibility of reaching these points. Note that REACHESNEW implies REACHESCHEAPER since the cost of any feasible path is finite, but REACHESCHEAPER does not imply REACHES-NEW.

Sampling-based Approximation. The simplest and fastest approach to approximating REACHESCHEAPER and REACHESNEW queries is to check whether sample points $x \in \mathcal{X}_{\mathbf{v}_{\text{end}}}$ meet the conditions on x in eq. (2) and eq. (3) respectively, returning $\texttt{True}$ if any sampled point does. These approximations are not conservative: the candidate path not being dominated at some sampled point x is sufficient to conclude that the candidate path is not dominated, but not necessary. For example, in fig. 4 (d), sparse sampling may miss the green interval where the cost-to-come of the candidate path $\tilde{g}(\mathbf{v}, x)$ is lower than the others, leading REACHESCHEAPER to return $\texttt{False}$, signaling that $\mathbf{v}$ is dominated, when in fact it is not. However, completeness and optimality are approached in the limit of infinite samples; in this sense we say GCS* is probabilistically complete and asymptotically optimal. We discuss this further in sec. 4.3.

Polyhedral-containment-based Approximation. We prove in appendix A that conservative (necessary but not sufficient) approximations of REACHES-NEW and REACHESCHEAPER enable completeness and optimality guarantees. We formulate such checks via polyhedral containment queries. One source of conservatism is that, unfortunately, these queries can check only *single domination*: They compare candidate path $\mathbf{v}$ to individual alternate paths in $S[\mathbf{v}_{\text{end}}]$. These checks can detect reachability domination like in fig. 4 (a) and (c) and cost domination like in fig. 4 (a), in which $\mathbf{v}$ is dominated by a single other path $\mathbf{v}^{(i)}$. These checks cannot detect reachability domination like in (d) or (e), or cost domination like in (e), in which the candidate path $\mathbf{v}$ is dominated by several alternate paths $\mathbf{v}^{(i)}$ collectively but not by any individual alternate path $\mathbf{v}^{(i)}$. Further conservatism occurs due to inability to compute exact containment queries tractably.

We first re-define REACHESCHEAPER and REACHESNEW in terms of set containment, before relaxing these conditions to tractable containment queries. For a path $\mathbf{v}$, we define the set in which a trajectory must reside ($\mathcal{P}_{\mathbf{v}}$), the set reachable via path $\mathbf{v}$ ($\mathcal{S}_{\mathbf{v}}$), and the epigraph of the optimal cost-to-come ($\mathcal{C}_{\mathbf{v}}$):

$$\mathcal{P}_{\mathbf{v}} := \{\mathbf{x} \mid \mathbf{x}_v \in \mathcal{X}_v \ \forall\, v \in \mathbf{v}, \ (\mathbf{x}_u, \mathbf{x}_v) \in \mathcal{X}_{(u,v)} \ \forall\, (u,v) \in \mathcal{E}_{\mathbf{v}}\}. \tag{4a}$$

$$\mathcal{S}_{\mathbf{v}} := \{x \in \mathcal{X}_{\mathbf{v}_{\text{end}}} \mid \exists\, \mathbf{x} \in \mathcal{P}_{\mathbf{v}} \text{ s.t. } x = \mathbf{x}_{\text{end}}\} \subseteq \mathcal{X}_{\mathbf{v}_{\text{end}}}. \tag{4b}$$

$$\mathcal{C}_{\mathbf{v}} := \{(x,\, l) \in \mathcal{X}_{\mathbf{v}_{\text{end}}} \times \mathbb{R} \mid \exists\, \mathbf{x} \in \mathcal{P}_{\mathbf{v}} \text{ s.t. } x = \mathbf{x}_{\text{end}},\, l \geq \sum_{(u,v) \in \mathcal{E}_{\mathbf{v}}} c(\mathbf{x}_u, \mathbf{x}_v)\}. \tag{4c}$$

We can now define the domination queries as set containment queries:

$$\text{REACHESCHEAPER}(\mathbf{v}, S[\mathbf{v}_{\text{end}}]) = \left(\mathcal{C}_{\mathbf{v}} \nsubseteq \bigcup_{\mathbf{v}' \in S[\mathbf{v}_{\text{end}}]} \mathcal{C}_{\mathbf{v}'} \right), \tag{5a}$$

$$\text{REACHESNEW}(\mathbf{v}, S[\mathbf{v}_{\text{end}}]) = \left(\mathcal{S}_{\mathbf{v}} \not\subseteq \bigcup_{\mathbf{v}' \in S[\mathbf{v}_{\text{end}}]} \mathcal{S}_{\mathbf{v}'} \right). \tag{5b}$$

The union over convex sets is not generally itself convex, and we know of no efficient way to compute these queries. Instead, we check single domination, comparing to individual other paths. In particular, we use

$$\nexists \mathbf{v}' \in S[\mathbf{v}_{\text{end}}] \text{ s.t. } \mathcal{C}_{\mathbf{v}} \subseteq \mathcal{C}_{\mathbf{v}'}, \quad \text{(6a)} \qquad \nexists \mathbf{v}' \in S[\mathbf{v}_{\text{end}}] \text{ s.t. } \mathcal{S}_{\mathbf{v}} \subseteq \mathcal{S}_{\mathbf{v}'}, \quad \text{(6b)}$$

as necessary conditions for REACHESCHEAPER (5a) and REACHESNEW (5b) respectively: if the candidate path is not dominated by the collective of other paths, then it certainly is not dominated by any individual other path.

These single domination checks are still non-trivial because they involve comparing projections of convex sets, which are expensive to compute explicitly. In our examples, vertex sets are polytopic and costs are linear, which allows us to use Sadraddini and Tedrake's sufficient condition for containment of affine transformations of polyhedra [23], evaluated by solving a convex program. Using this condition to check (6b) and (6a), NOTDOMINATED($\mathbf{v}, S[\mathbf{v}_{\text{end}}]$) is conservative: If candidate path $\mathbf{v}$ is not dominated, it certainly returns `True`, but if candidate path $\mathbf{v}$ is dominated, it may return `True` or `False`. In the case of more general convex vertex sets and costs, Jones and Morari's work on computing inner and outer polytopic approximations of convex sets [10] can be used before evaluating these containment queries.

4.3   Properties of the Algorithm

We state properties of the algorithm in this section, and include proofs in appendix A. In particular, under conservative REACHESCHEAPER or REACHESNEW checks, GCS* is complete; and under conservative REACHESCHEAPER checks, GCS* is cost optimal:

Theorem 1. *GCS* returns a path from s to t in finite iterations if one exists.*

Theorem 2. *GCS* is **cost optimal**: GCS* returns an optimal trajectory from s to t in finite iterations if one exists.*

Note that, like A* [22], GCS* as stated cannot be assured to terminate in finite iterations in the case where prog. (1) is infeasible and the graph is infinite or cycles are permitted. GCS* is made complete in the infeasible case by adding a limit on path length to prog. (1) and placing a corresponding upper limit on the lengths of paths that get added to S and Q in lines 11 and 12.

Efficiency. In contrast to A*, GCS* is not optimally efficient. That is, given the heuristic information $\tilde{h}$, it can be shown that GCS* expands subpaths that may not be expanded by some other hypothetical algorithm guaranteed to return the optimal solution. Conceptually, GCS* loses this property that applies to A* because when GCS* expands a subpath $\mathbf{v}$, by considering paths through the children of $\mathbf{v}_{\text{end}}$, GCS* now has access to potentially improved lower-bound on

the true cost-to-go, $\tilde{h}(x)$, for x in the reachable set of $\mathbf{v}$, $\mathcal{S}_{\mathbf{v}}$ (4b), but we do not choose to use this information to update $\tilde{h}(x)$. A* with a consistent heuristic does not suffer from this inefficiency because it never expands a node twice [4], whereas GCS* may expand many subpaths whose terminal sets contain points in $\mathcal{S}_{\mathbf{v}}$.

Properties when using sampling-based domination checks. While we have proven that GCS* is cost-optimal and complete under conservative IS-DOMINATED checks, a sampling-based implementation is sometimes preferred, as discussed in sec. 4.2. In the limit of infinite samples, the sampling-based implementations of NOTDOMINATED are exact. Thus it follows from theorems 1 and 2 that sampling-based GCS* is probabilistically complete and asymptotically cost-optimal. One might raise concern that the backward-reachable set from the target in some set $\mathcal{X}_v$ may never be sampled, leading some important feasible path to be pruned. However, because all $\mathcal{P}_{\mathbf{v}}$ (4a) are compact, the sets being checked for containment, $\mathcal{S}_{\mathbf{v}}$ (4b), are closed. Therefore, $\mathcal{S}_{\mathbf{v}} \backslash \bigcup_{\mathbf{v}' \in S[\mathbf{v}_{\mathrm{end}}]} \mathcal{S}_{\mathbf{v}'}$ with respect to the topology of $\mathcal{S}_{\mathbf{v}}$ (the space being sampled) is open and thus either empty or positive-measure. If it has positive measure, it will eventually be sampled.

5   Application to Planar Pushing

Tasks involving making and breaking contact while constraining non-penetration notoriously lead to an explosion of discrete modes, especially as the number of bodies scales. Indeed, the naive formulation in sec. 5.1 sees this combinatorial growth. With more careful construction of contact modes [9], or by introducing hierarchy [1], one could greatly reduce search space for these problems. However, additional structure can make such methods less general. Our goal with these experiments is not to demonstrate state of the art performance on contact-rich manipulation tasks specifically, but instead to show that our method is able to approach such large problems.

5.1   Formulation

We implement a simplified planar pushing model with polyhedral robots and objects that can translate but not rotate, and static obstacles. Robot-object and object-object contact are frictionless, and motion quasi-static, where bodies' velocities are proportional to the net forces applied to them. We enforce non-penetration between all pairs of bodies, defining collision-free sets over robot and object positions. We also define sets over positions and contact forces defining the permissible physical contact behavior for each pair of bodies. Together, these two kinds of discrete modes define the sets of the graph G. Each point $\mathbf{x}_v \in \mathcal{X}_v$ in prog. 1 is comprised of n_k knot points, defining the planar positions for each robot and object, actuation forces for each robot, and a force magnitude for each pair of bodies in contact. The constraints defining these sets, which we explain throughout this section, are straightforward to compose. As such, the SUCCESSORS operator can construct them on demand. In particular, the

successors of some vertex are defined by changing the contact state or the non-penetration separating hyperplane (defined below) for a single pair of bodies.

All constraints are linear, leading to polyhedral sets. However, extensions of this model to include rotations could involve semidefinite relaxations, leading to spectrahedral sets, in a formulation similar to that of Graesdal et al. [6]. This work could also be extended to borrow other components of the GCS formulation of planar pushing from Graesdal et al., like frictional contact. We omit these features for simplicity. However, the assumption that bodies are polyhedral is harder to relax, as it allows for an exact polyhedral decomposition of collision-free space. Alternative settings that require only an approximate convex decomposition of the free configuration space can handle rotations and non-polyhedral bodies [20,27,3]. However, these formulations are not conducive to making contact due to incomplete coverage. As these decompositions are generally composed offline and cannot produce new sets as quickly as our Successors operator can, this formulation would either require building the entire graph before running GCS* or accepting slower evaluation of Successors.

Non-penetration. We constrain that each pair of bodies do not penetrate one another by imposing that one face of one of the two polyhedra is a separating hyperplane: all vertices of one body lie on one side, and all vertices of the other body lie on the other side.

Contact. Every pair of bodies may have any of the following kinds of contact: no contact, any face of one body in contact with any face of another, or any face of one body in contact with any vertex of another. Gathering these options over all pairs of bodies produces a *contact set*. Each pair of bodies in contact in each contact set is accompanied by a force magnitude variable, leading to variation in dimension between sets. This force acts normal to the face in contact. Position constraints are added to ensure the associated contact is physically feasible. Relative sliding of features in contact is allowed within a single contact set, as long as they remain in contact.

Quasi-static dynamics. For each object and robot, we enforce that between consecutive knot points, translation is proportional to the sum of forces on the body, including a robot's actuation force. Edge constraints enforce that robot and object positions remain the same across edges – for an edge (u, v) in eq. (1), the positions from the last knot point in $\mathbf{x}_u$ must equal the positions from the first knot point in $\mathbf{x}_v$. As a result of this constraint, a path containing the edge (u, v) is only feasible if there is a shared position that is feasible in both convex sets associated with these vertices, $\mathcal{X}_u$ and $\mathcal{X}_v$.

5.2   Implementation Details

We use $n_k = 2$ knot points per set. Our edge cost is the L_1 norm distance between knot points, plus a constant penalty for switching modes. That is, for $v \neq t$, $c(\mathbf{x}_u, \mathbf{x}_v) := 1 + \sum_i w_i ||p_{v,1}^i - p_{v,0}^i||_1$, for weights w_i, where $p_{v,j}^i$ gives the position of the ith moving body (object or robot) at the jth knot point in vertex

v. $c(\mathbf{x}_u, t) = 0$ (no movement in the target set). We use $w_i = 1$ for all i. Our sampling-based implementations can handle L_2 norm or L_2 norm squared edge costs easily. However, as discussed in sec. 4.2, our containment-based implementations need modification to accommodate non-linear costs.

For simplicity, our nominal $\tilde{h}$ is the cost of a "shortcut" edge between the point x and some point in the target set $x_t \in \mathcal{X}_t$ chosen to minimize $\tilde{h}$. If an edge does exist directly to the target, the true cost of that edge is used. Otherwise, $\tilde{h}(x) \leftarrow 1 + c(x, x_t)$, using a weight of 0.2 for robot displacements instead of the true weights $w_i = 1$. Because the weights do not exceed the true weights, this yields an admissible heuristic. Note that 1 is added to $c(x, x_t)$ because we know that a constant penalty of 1 is applied for every mode switch and there will be at least one additional mode switch before reaching the target. For speed, we also conduct experiments with an ϵ-*suboptimal* heuristic instead, where we scale $\tilde{h}$ by some ϵ, $\tilde{h}(x) \leftarrow \epsilon \cdot (1 + c(x, x_t))$, to prune more aggressively, using $\epsilon = 10$. The shortcut edge ignores contact dynamics, making this heuristic less informative in some scenarios than others. $\tilde{h}$ can be further optimized, but this is not the focus of this paper. Practitioners may apply domain knowledge to formulate informative heuristics, trading off between informativeness and computational cost.

For sampling-based domination checks, we use only a single sample per check. As we will discuss, empirically, this is sufficient. In practice, due to edge constraints, the reachable set for a path $\mathbf{v}$, $\mathcal{S}_\mathbf{v} \subseteq \mathcal{X}_{\mathbf{v}_{\text{end}}}$, is often low-volume within the terminal set $\mathcal{X}_{\mathbf{v}_{\text{end}}}$. As such, we sample uniformly in $\mathcal{X}_{\mathbf{v}_{\text{end}}}$ and then project onto $\mathcal{S}_\mathbf{v}$.

For containment-based methods, we first perform a cheap single-sample NOT-DOMINATED check, and only check containment if it returns `False`. Additionally, taking advantage of the unique structure of our planar pushing formulation, we only check the domination conditions on the last position knot point instead of all variables in the set. To reduce problem size for domination checks, we "solve away" equality constraints by parameterizing in their nullspace.

6   Results

Table 2 presents results for three planar pushing tasks: AROUND, SQUEEZE, and STACK shown in figs. 1 and 5. AROUND is the simplest task: its graph has 194 vertices and 8,328 edges. SQUEEZE has a larger graph with 628 vertices and 56,806 edges. The dimensions of sets in AROUND and SQUEEZE range from 4 to 12. STACK (fig. 1) has 5 bodies, leading to a combinatorial explosion of approximately 1.3×10^9 sets and up to 8.5×10^{17} edges. The dimensions of its sets range from 8 to 24. Due to the sizes of the graphs, the direct convex optimization approach [15] cannot be used, thus we compare GCS* against IxG and IxG* [18], the state-of-the-art for incrementally solving SPP on GCS.

For sampling-based domination checks, as we increase the number of samples per check, we approach completeness and optimality. Due to the curse of dimensionality, such coverage demands exponentially more samples as set dimensions increase. However, in our tasks, we found that very sparse sampling – in particular, a single sample per domination check – still resulted in good solution quality. This is an interesting result that enables an efficient and de facto
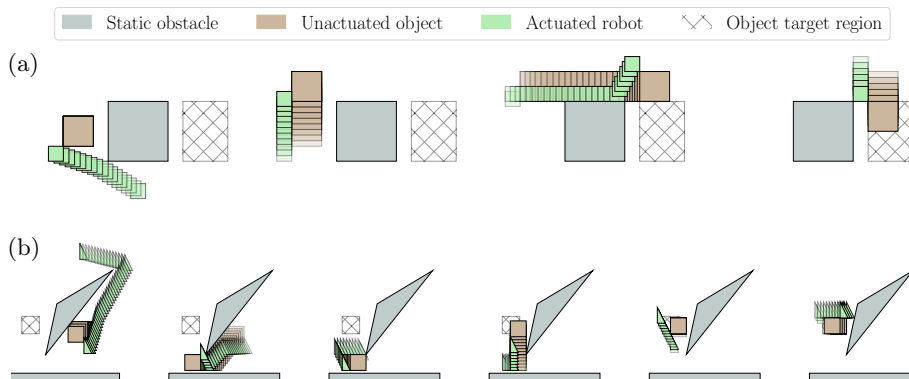
Fig. 5: The ϵ-suboptimal solution trajectories found by GCS* using the sampling-based REACH-ESCHEAPER domination check on the AROUND Task (a) and SQUEEZE Task (b). In SQUEEZE, in the second frame, the robot pushes the object past the obstacle to make room for itself to maneuver around in the third frame. In the fourth, the robot uses contact with the obstacle to slide the object vertically until the robot can fit underneath it. Videos of these trajectories are available on the project website.

nearly optimal implementation. We understand this to be a property particular to our problem structure, in which the condition that two paths reach the same final knot point within the same vertex is quite restrictive. For example, the sampling-based REACHESNEW approximation asks whether any path in S can achieve a trajectory ending at a particular sampled object position and robot position in some contact state that can be achieved by a candidate path, calling the candidate path dominated if so. In this event, it is likely that that path in S can reach much of what the candidate path can, and the candidate path can be culled with little consequence.

Containment-based domination checks scale poorly with path length, or, more precisely, the dimension of the full trajectory through it. This is reflected in the solve times in the rows marked "Cont." in table 2.

The number of paths expanded is impacted greatly by the strength of the heuristic in the context of the particular task. In general, the "shortcut" edge heuristic is strong for AROUND and STACK and weaker for SQUEEZE which requires squeezing through a narrow gap then maneuvering around the object to push it from a different direction multiple times. As expected, use of an inadmissible heuristic leads to cost increase. Typically, containment-based checks lead to more paths being expanded as compared to sampling-based checks (as seen in table 2) because of false negatives from sampling, as well as conservatism of the single domination criteria. However, containment-based checks could also lead to fewer paths being expanded because important paths are not wrongly pruned, as they might be when using sampling-based checks (not observed in table 2).

In comparison to our methods, the baselines require significant time to compute heuristic values for the entire graph before starting their query phase. While this approach can yield a stronger heuristic (as in the bottommost row, where IxG expands fewer paths than GCS* for AROUND), for these tasks the solve

Table 2: Here we show the solve time, solution cost, and number of expanded paths (Exp) for three planar pushing tasks. Our algorithm GCS* is compared to the baselines IxG* and IxG (grey). Sampling (Sampl.) and containment (Cont.) based implementations (Impl.) of the domination checkers (DC) ReachesCheaper (RC) and ReachesNew (RN) are shown. A dagger (†) indicates that the property holds probabilistically or asymptotically. Some experiments returned infeasible (`Fail`), were stopped after 10h (×), or could not be run as the graph could not be explicitly constructed (-). The baselines required preprocessing times of 1.1m and 1.7h for AROUND and SQUEEZE respectively. Our methods require no preprocessing time. All computation was performed on an AMD Ryzen 9 7950x 16-core 32-thread processor.

Alg.	DC	Impl.	Heur.	Com-plete	Cost optimal	AROUND			SQUEEZE			STACK		
						Time	Cost	Exp	Time	Cost	Exp	Time	Cost	Exp
GCS*	RC	Sampl.	Admiss.	Yes†	Opt.†	26.2s	27.5	432	4m	47.80	2114	×	×	×
GCS*	RC	Cont.	Admiss.	Yes	Opt.	7.5h	27.5	734	×	×	×	×	×	×
IxG*	N.A.	N.A.	Admiss.	Yes	Opt.	×	×	×	×	×	×	-	-	-
GCS*	RC	Sampl.	Inadm.	Yes†	ϵ-subopt.†	2.9s	27.5	51	38.3s	54.55	381	21.9s	67.53	57
GCS*	RC	Cont.	Inadm.	Yes	ϵ-subopt.	4.6m	27.5	73	×	×	×	×	×	×
IxG*	N.A.	N.A.	Inadm.	Yes	ϵ-subopt.	×	×	×	×	×	×	-	-	-
GCS*	RN	Sampl.	Inadm.	Yes†	Subopt.	2.5s	27.5	44	20s	57.42	259	21.8s	67.53	57
GCS*	RN	Cont.	Inadm.	Yes	Subopt.	35.4s	28.5	75	20.5m	57.42	458	3.5m	67.53	64
IxG	N.A.	N.A.	Inadm.	No	Subopt.	4.7s	28	31	2.3m	Fail	615	-	-	-

times can be dominated by this pre-processing phase. When the graph is very large, this pre-processing can become intractable (as in STACK). Because IxG* prunes paths based on a global upper bound found by IxG, in cases where IxG is unable to return a solution (as in SQUEEZE), or simply when an admissible heuristic is used (as in row 3), no paths will be pruned, resulting in intractably large search spaces. As GCS* is complete, and our domination checks do not rely on a global upper bound, GCS* does not experience these particular limitations.

7 Conclusion

We propose GCS*, a forward heuristic search algorithm for solving large discrete-continuous planning problems formulated as GCS. We define two domination checks ReachesNew and ReachesCheaper, as well as containment and sampling-based implementations of those checks that allow GCS* to be complete and optimal, or have probabilistic/asymptotic versions of those properties, respectively. GCS* provides a principled way of addressing the challenges of applying graph search to the discrete-continuous setting. We demonstrate settings in which GCS* performs favourably compared to the state-of-the-art. For application to real-world planar pushing tasks, further work would be required to handle rotations [6]. Incorporating these insights into algorithms that leverage hierarchy, factorization or learned heuristics could solve more complex problems faster.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Cheng, X., Patil, S., Temel, Z., Kroemer, O., Mason, M.T.: Enhancing dexterity in robotic manipulation via hierarchical contact exploration **9**(1), 390–397. https://doi.org/10.1109/LRA.2023.3333699, https://ieeexplore.ieee.org/abstract/document/10319720

2. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms. The MIT Press, 4th edn.

3. Dai, H., Amice, A., Werner, P., Zhang, A., Tedrake, R.: Certified polyhedral decompositions of collision-free configuration space . https://doi.org/10.1177/02783649231201437

4. Dechter, R., Pearl, J.: Generalized best-first search strategies and the optimality of a* **32**(3), 505–536. https://doi.org/10.1145/3828.3830, https://dl.acm.org/doi/10.1145/3828.3830

5. Garrett, C.R., Chitnis, R., Holladay, R., Kim, B., Silver, T., Kaelbling, L.P., Lozano-Pérez, T.: Integrated task and motion planning **4**, 265–293. https://doi.org/10.1146/annurev-control-091420-084139, https://www.annualreviews.org/content/journals/10.1146/annurev-control-091420-084139

6. Graesdal, B.P., Chew Chia, S.Y., Marcucci, T., Morozov, S., Amice, A., Parrilo, P.A., Tedrake, R.: Towards tight convex relaxations for contact-rich manipulation. https://doi.org/10.48550/arXiv.2402.10312, http://arxiv.org/abs/2402.10312

7. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths **4**(2), 100–107. https://doi.org/10.1109/TSSC.1968.300136, https://ieeexplore.ieee.org/document/4082128, conference Name: IEEE Transactions on Systems Science and Cybernetics

8. Hartmann, V.N., Orthey, A., Driess, D., Oguz, O.S., Toussaint, M.: Long-horizon multi-robot rearrangement planning for construction assembly **39**(1), 239–252. https://doi.org/10.1109/TRO.2022.3198020, https://ieeexplore.ieee.org/document/9868234/

9. Huang, E., Cheng, X., Mason, M.T.: Efficient contact mode enumeration in 3d. vol. 17, pp. 485–501. Springer International Publishing. https://doi.org/10.1007/978-3-030-66723-8_29, http://link.springer.com/10.1007/978-3-030-66723-8_29, book Title: Algorithmic Foundations of Robotics XIV Series Title: Springer Proceedings in Advanced Robotics

10. Jones, C.N., Morari, M.: Polytopic approximation of explicit model predictive controllers **55**(11), 2542–2553. https://doi.org/10.1109/TAC.2010.2047437, http://ieeexplore.ieee.org/document/5617688/

11. Kavraki, L., Svestka, P., Latombe, J.C., Overmars, M.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces **12**(4), 566–580. https://doi.org/10.1109/70.508439, https://ieeexplore.ieee.org/document/508439, conference Name: IEEE Transactions on Robotics and Automation

12. LaValle, S.M.: Rapidly-exploring random trees: A new tool for path planning, https://msl.cs.uiuc.edu/~lavalle/papers/Lav98c.pdf

13. Mao, J., Tenenbaum, J.B., Lozano-Perez, T., Kaelbling, L.P.: Learning reusable manipulation strategies. In: 7th Annual Conference on Robot Learning. https://openreview.net/forum?id=ihqTtzS83VS

14. Marcucci, T., Petersen, M., von Wrangel, D., Tedrake, R.: Motion planning around obstacles with convex optimization **8**(84), eadf7843. https://doi.org/10.1126/scirobotics.adf7843, https://www.science.org/doi/10.1126/scirobotics.adf7843

15. Marcucci, T., Umenberger, J., Parrilo, P., Tedrake, R.: Shortest paths in graphs of convex sets **34**(1), 507–532. https://doi.org/10.1137/22M1523790, https://epubs.siam.org/doi/full/10.1137/22M1523790

16. Morozov, S., Marcucci, T., Amice, A., Graesdal, B.P., Bosworth, R., Parrilo, P., Tedrake, R.: Multi-query shortest-path problem in graphs of convex sets. In: Proceedings of the Sixteenth Workshop on the Algorithmic Foundations of Robotics

17. Natarajan, R., Choset, H., Likhachev, M.: Interleaving graph search and trajectory optimization for aggressive quadrotor flight **6**(3), 5357–5364. https://doi.org/10.1109/LRA.2021.3067298, http://arxiv.org/abs/2101.12548

18. Natarajan, R., Liu, C., Choset, H., Likhachev, M.: Implicit graph search for planning on graphs of convex sets. In: Proceedings of Robotics: Science and Systems 2024

19. Pang, T., Suh, H.J.T., Yang, L., Tedrake, R.: Global planning for contact-rich manipulation via local smoothing of quasi-dynamic contact models **39**(6), 4691–4711. https://doi.org/10.1109/TRO.2023.3300230, https://ieeexplore.ieee.org/document/10225433, conference Name: IEEE Transactions on Robotics

20. Petersen, M., Tedrake, R.: Growing convex collision-free regions in configuration space using nonlinear programming. https://doi.org/10.48550/arXiv.2303.14737, http://arxiv.org/abs/2303.14737

21. Pohl, I.: Heuristic search viewed as path finding in a graph **1**(3), 193–204. https://doi.org/10.1016/0004-3702(70)90007-X, https://linkinghub.elsevier.com/retrieve/pii/000437027090007X

22. Russell, S.J., Norvig, P., Chang, M.W., Devlin, J., Dragan, A., Forsyth, D., Goodfellow, I., Malik, J., Mansinghka, V., Pearl, J., Wooldridge, M.J.: Artificial intelligence: a modern approach. Pearson series in artificial intelligence, Pearson, 4th edn., HOLLIS number: 99154334206503941

23. Sadraddini, S., Tedrake, R.: Linear encodings for polytope containment problems. In: 2019 IEEE 58th Conference on Decision and Control (CDC). pp. 4367–4372. IEEE. https://doi.org/10.48550/arXiv.1903.05214, http://arxiv.org/abs/1903.05214

24. Sundar, K., Rathinam, S.: a^* for graphs of convex sets. https://doi.org/10.48550/arXiv.2407.17413, http://arxiv.org/abs/2407.17413

25. Toussaint, M.: Logic-geometric programming: An optimization-based approach to combined task and motion planning. In: Proceedings of the 24th International Conference on Artificial Intelligence. pp. 1930–1936. IJCAI'15, AAAI Press

26. Toussaint, M., Lopes, M.: Multi-bound tree search for logic-geometric programming in cooperative manipulation domains. In: 2017 IEEE International Conference on Robotics and Automation (ICRA). pp. 4044–4051. IEEE. https://doi.org/10.1109/ICRA.2017.7989464, http://ieeexplore.ieee.org/document/7989464/

27. Werner, P., Amice, A., Marcucci, T., Rus, D., Tedrake, R.: Approximating robot configuration spaces with few convex sets using clique covers of visibility graphs. https://doi.org/10.48550/arXiv.2310.02875, http://arxiv.org/abs/2310.02875, version: 2

A   Proofs of Algorithmic properties

We prove that, under conservative NOTDOMINATED or specifically REACH-ESCHEAPER checks, GCS* is complete or optimal, respectively: The proofs rely on these checks being necessary but not sufficient conditions for eq. (3) and eq. (2). Optimality also relies on $\tilde{h}$ being pointwise admissible (def. 1).

Completeness. We show that if prog. (1) is feasible, GCS* returns a feasible path in finite iterations. However, like A* [22], GCS* as stated cannot be assured to terminate in finite iterations in the case where prog. (1) is infeasible and the graph is infinite or cycles are permitted. GCS* is made complete in the infeasible case by adding a limit on path length to prog. (1) and placing an upper limit on the lengths of paths that get added to S and Q in lines 11 and 12.

Lemma 1. *For any path $\mathbf{v}$ in Q, GCS* either expands $\mathbf{v}$ after a finite number of iterations or returns a feasible path to the target first.*

Proof. Path $\mathbf{v}$ has some finite estimated total cost $\tilde{f}(\mathbf{v})$. Because all edge costs are positive and bounded away from zero, there exist a finite number of paths $\mathbf{v}'$ with cost-to-come that do not exceed $\tilde{f}(\mathbf{v})$, $\tilde{g}(\mathbf{v}') \leq \tilde{f}(\mathbf{v})$. As $\tilde{f}(\mathbf{v}') \geq \tilde{g}(\mathbf{v}')$ for any $\mathbf{v}'$, there are a finite number of paths $\mathbf{v}'$ ever added to Q such that $\tilde{f}(\mathbf{v}') \leq \tilde{f}(\mathbf{v})$. As GCS* always expands the path in Q with the lowest $\tilde{f}$ value, GCS* will eventually exhaust all such paths and expand $\mathbf{v}$, unless it terminates with a feasible path to the target first.

Lemma 2. *If target t is reachable from source s, then some path that reaches it is added to Q in finite iterations.*

Proof. Suppose, for contradiction, that t is reachable from s, but no path that reaches it is added to Q in finite iterations. There is some feasible path $\mathbf{v}$ where $\mathbf{v}_{\text{start}} = s$, $\mathbf{v}_{\text{end}} = t$, and some feasible trajectory $\mathbf{x} \in \mathcal{P}_{\mathbf{v}}$. Let j be the largest index such that there exists a path $\mathbf{v}'$ (which may or may not be a subpath of $\mathbf{v}$) terminating in $\mathbf{v}_j$ ($\mathbf{v}'_{\text{end}} = \mathbf{v}_j$) and reaching $\mathbf{x}_j$, that ever gets added to Q. Such a $j \geq 0$ exists because Q is initialized with the path containing just s. By lemma 1, $\mathbf{v}'$ is expanded in finite iterations, unless a path reaching t is returned first. If a path reaching t is returned first, we achieve contradiction trivially. Consider instead the case where $\mathbf{v}'$ is expanded in finite iterations. By assumption, $\mathbf{v}_{j+1}$ is a successor of $\mathbf{v}_j$, and so the extended path $[\mathbf{v}', \mathbf{v}_{j+1}]$ must be added to Q when $\mathbf{v}'$ is expanded. We can be sure it is added to Q instead of being rejected by NOTDOMINATED in line 10 because $(\mathbf{x}_j, \mathbf{x}_{j+1}) \in \mathcal{X}_{(\mathbf{v}_j, \mathbf{v}_{j+1})}$, making $\mathbf{x}_{j+1}$ reachable via the extended path $[\mathbf{v}', \mathbf{v}_{j+1}]$, and, by assumption, no path terminating in $\mathbf{v}_{j+1}$ and reaching $\mathbf{x}_{j+1}$ has been added to Q prior, and thus nor has it been added to $S[\mathbf{v}_{j+1}]$, as S is only added to in conjunction with Q. However, the extended path $[\mathbf{v}', \mathbf{v}_{j+1}]$ is now a path in Q terminating in $\mathbf{v}_{j+1}$ and reaching $\mathbf{x}_{j+1}$, creating a contradiction with the definition of j.

Theorem 1. *GCS* returns a path from s to t in finite iterations if one exists.*

Proof. By lemma 2, a feasible path from s to t is added to Q in finite iterations if one exists. By lemma 1, either this path is expanded in finite iterations (in which case it is immediately returned), or another such feasible path from s to t is returned first. In both cases, a path from s to t is returned in finite iterations.

Cost-optimality. In this section, we assume NOTDOMINATED is an exact or conservative check for REACHESCHEAPER.

Lemma 3. *Suppose prog. (1) is feasible and* $\mathbf{v}$ *is an optimal path from* s *to* t. *If* t *is unexpanded, there is a path* $\mathbf{v}'$ *in* Q *with* $\mathbf{v}'_{end} = \mathbf{v}_i$ *for some* i, *such that* $\mathbf{v}'$ *attains the optimal cost-to-come to* $\mathbf{x}^*(\mathbf{v})_i$, $\tilde{g}(\mathbf{v}', \mathbf{x}^*(\mathbf{v})_i) = g^*(\mathbf{x}^*(\mathbf{v})_i)$.

Proof. If GCS* has not completed one iteration, the lemma holds trivially via the path containing only s, $\mathbf{v}^s := [s] \in Q$. Suppose instead $\mathbf{v}^s$ has been expanded. Let $\mathbf{v}$ be any optimal path from s to t, and let $\mathbf{x} := \mathbf{x}^*(\mathbf{v})$ be an optimal trajectory through $\mathbf{v}$. Let i be the largest index for which there exists an expanded path $\mathbf{v}'$ such that $\mathbf{v}_i = \mathbf{v}'_{end}$, with optimal cost-to-come to $\mathbf{x}_i$, $\tilde{g}(\mathbf{v}', \mathbf{x}_i) = g^*(\mathbf{x}_i)$. That is, the ith vertex on $\mathbf{v}$ has been expanded on some path $\mathbf{v}'$ which may or may not be a subpath of $\mathbf{v}$ but attains the same (optimal) cost-to-come to $\mathbf{x}_i$. Since s has been expanded, and t has not, $i \in [0, end - 1]$. Because $(\mathbf{x}_i, \mathbf{x}_{i+1}) \in \mathcal{X}_{(\mathbf{v}_i, \mathbf{v}_{i+1})}$ (we know this because the optimal trajectory $\mathbf{x}$ traverses this edge via these points), the extended path $[\mathbf{v}', \mathbf{v}_{i+1}]$ reaches $\mathbf{x}_{i+1}$ with optimal cost-to-come $g^*(\mathbf{x}_{i+1}) = g^*(\mathbf{x}_i) + c(\mathbf{x}_i, \mathbf{x}_{i+1})$.

We now argue that once $\mathbf{v}'$ was expanded, there must have been, at some point present or prior, a path added to Q terminating at $\mathbf{v}_{i+1}$ and reaching $\mathbf{x}_{i+1}$ with optimal cost-to-come. Then, we argue that it is still in Q. When $\mathbf{v}'$ was expanded, the extended path $[\mathbf{v}', \mathbf{v}_{i+1}]$ was placed on Q unless it was rejected by REACHESCHEAPER returning `False` in line 10. If REACHESCHEAPER returned `False`, then another path was already in $S[\mathbf{v}_{i+1}]$ that terminated at $\mathbf{v}_{i+1}$ and reached $\mathbf{x}_{i+1}$ with optimal cost-to-come. Paths are added to S only when added to Q, so this path was added to Q. By the definition of i, whichever such path was at any point in Q has not been expanded and is still in Q.

Corollary 1. *Suppose prog. (1) is feasible, and* $\mathbf{v}$ *is an optimal path from* s *to* t. *If alg. 2 has not terminated, there is some* $\mathbf{v}'$ *in* Q *such that* $\mathbf{v}'_{end} \in \mathbf{v}$ *and the estimated total cost for* $\mathbf{v}'$ *does not exceed the true optimal cost,* $\tilde{f}(\mathbf{v}') \leq f^*(s)$.

Proof. As alg. 2 terminates only when t is expanded, t is unexpanded under these assumptions. By lemma 3, there is an unexpanded search path $\mathbf{v}'$ in Q with $\mathbf{v}'_{end} = \mathbf{v}_i$ for some i and an optimal cost-to-come to $\mathbf{x}^*(\mathbf{v})_i$.

We know $\tilde{f}(\mathbf{v}') \leq \tilde{f}(\mathbf{v}', \mathbf{x}^*(\mathbf{v})_i)$ because, more generally speaking, a path constrained to go through $\mathbf{v}'$ is estimated to be at least as cheap as a path additionally constrained to go through any particular point. Then,

$$
\begin{aligned}
\tilde{f}(\mathbf{v}', \mathbf{x}^*(\mathbf{v})_i) &= \tilde{g}(\mathbf{v}', \mathbf{x}^*(\mathbf{v})_i) + \tilde{h}(\mathbf{x}^*(\mathbf{v})_i) \quad \text{(By definition of } \tilde{f}) \\
&= g^*(\mathbf{x}^*(\mathbf{v})_i) + \tilde{h}(\mathbf{x}^*(\mathbf{v})_i) \quad \text{(By lemma 3)} \\
&\leq g^*(\mathbf{x}^*(\mathbf{v})_i) + h^*(\mathbf{x}^*(\mathbf{v})_i) \quad \text{(By } \tilde{h} \text{ pointwise admissibility)} \\
&= f^*(s) \quad \text{(Since } \mathbf{x}^*(\mathbf{v})_i \text{ is on an optimal trajectory).}
\end{aligned}
$$

Thus, we have $\tilde{f}(\mathbf{v}') \leq f^*(s)$.

Theorem 2. *GCS* is **cost optimal**: GCS* returns an optimal trajectory from s to t in finite iterations if one exists.*

Proof. By theorem 1, GCS* returns a feasible path $\mathbf{v}'$ from s to t in finite iterations. Suppose for contradiction that the cost-to-come to t through $\mathbf{v}'$ is suboptimal, $\tilde{g}(\mathbf{v}', \mathbf{x}^*(\mathbf{v}')_{\text{end}}) > f^*(s)$. By admissibility, $h^*(x) = \tilde{h}(x) = 0 \ \forall x \in \mathcal{X}_t$. Thus, $\tilde{f}(\mathbf{v}') = \tilde{g}(\mathbf{v}', \mathbf{x}^*(\mathbf{v}')_{\text{end}}) > f^*(s)$. By corollary 1, just before termination, there existed a path $\mathbf{v}$ in Q with $\tilde{f}(\mathbf{v}) \leq f^*(s) < \tilde{f}(\mathbf{v}')$. However, alg. 2 always pops the path in Q with the lowest $\tilde{f}$ value, and returns a path only after popping it. This contradicts the assumption that $\mathbf{v}'$ was returned.