Robotics: Science and Systems 2024
Delft, Netherlands, July 15-July 19, 2024

Implicit Graph Search for Planning on Graphs of Convex Sets

Ramkumar Natarajan*, Chaoqi Liu†, Howie Choset*, Maxim Likhachev*
*The Robotics Institute at Carnegie Mellon University
{rnataraj, choset, maxim}@cs.cmu.edu
†Department of Computer Science at the University of Illinois at Urbana-Champaign
chaoqil2@illinois.edu
https://sbpl.github.io/ixg/

Fig. 1: Smooth, collision-free motions generated by INSATxGCS (IxG) for various tasks with three Motoman HC10DTP arms operating simultaneously in a 18-DoF multi-arm assembly scenario.

***Abstract*—Smooth, collision-free motion planning is a fundamental challenge in robotics with a wide range of applications such as automated manufacturing, search & rescue, underwater exploration, etc. Graphs of Convex Sets (GCS) is a recent method for synthesizing smooth trajectories by decomposing the planning space into convex sets, forming a graph to encode the adjacency relationships within the decomposition, and then simultaneously searching this graph and optimizing parts of the trajectory to obtain the final trajectory. To do this, one must solve a Mixed Integer Convex Program (MICP) and to mitigate computational time, GCS proposes a convex relaxation that is empirically very tight. Despite this tight relaxation, motion planning with GCS for real-world robotics problems translates to solving the simultaneous batch optimization problem that may contain millions of constraints and therefore can be slow. This is further exacerbated by the fact that the size of the GCS problem is invariant to the planning query. Motivated by the observation that the trajectory solution lies only on a fraction of the set of convex sets, we present two implicit graph search methods for planning on the graph of convex sets called INSATxGCS (IxG) and IxG*. INterleaved Search And Trajectory optimization (INSAT) is a previously developed algorithm that alternates between searching on a graph and optimizing partial paths to find a smooth trajectory. By using an implicit graph search method INSAT on the graph of convex sets, we achieve faster planning while ensuring stronger guarantees on completeness and optimality. Moveover, introducing a search-based technique to plan on the graph of convex sets enables us to easily leverage well-established techniques such as search parallelization, lazy planning, anytime planning, and replanning as future work. Numerical comparisons against GCS demonstrate the superiority of IxG across several applications, including planning for an 18-degree-of-freedom multi-arm assembly scenario.**

I. Introduction

Trajectory optimization is widely used in motion planning in high-dimensional state spaces under kinodynamic constraints. However, complex configuration spaces cluttered with obstacles, as few as one, can make the optimization-based motion planning problem nonconvex [7, 31, 33, 38]. The prevailing wisdom in optimization is to somehow transform the problem into a convex program, which can be solved efficiently for very large problems. Recently, graphs of convex sets [1] [20] decomposed the planning space into convex sets,

[1]Per the authors of GCS [18, 20], the graph of convex sets is a directed graph in which each vertex is paired with a convex set whose spatial position is a continuous variable constrained to lie in the convex set and edge cost is a given convex function of the position of the vertices that this edge connects. Whereas, GCS is the technique to plan on the graph of convex sets. The terms "GCS" and "graph of convex sets" were used interchangeably in [20] but were clear from the context and we follow the same in this paper.

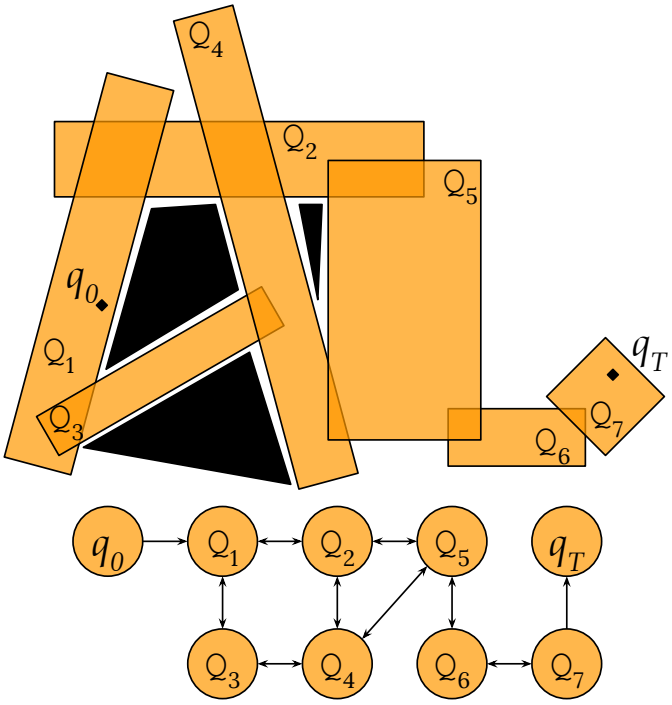


Fig. 2: An example of decomposition of free space into convex sets $\mathcal{Q}_1 \cup \ldots \cup \mathcal{Q}_n$ with start q_0 and goal q_T states (above) and their relationship represented as a graph (below).

and represented this decomposition as a graph (Fig. 2). This decomposition has the potential to represent the planning space we encounter in most of the robotics motion planning problems as a sparse graph that covers most of the configuration space. Using this representation, motion planning on the graphs of convex sets introduced in [20] called GCS is a remarkable formulation [18] that transforms an inherently nonconvex problem into a convex optimization. GCS proposed a joint batch optimization over the entire graph (Fig 3) to simultaneously choose the set of convex sets to traverse through and find the smooth trajectory via them. Such a giant optimization is conceivable as the research in convex optimization is very mature and is applied to efficiently solve large real-world problems. However, the joint batch optimization in GCS means that for any given task, the size of the GCS problem is independent of the planning query. It can potentially solve a massive optimization for every planning query. For example, for the challenging task of finding a trajectory to navigate through the clutter and finding a trajectory between two points that can be trivially connected in a large high-dimensional planning space, GCS attempts to solve the same-sized optimization program. This fundamentally limits the scope of its capability to plan fast for articulated systems, plan with a receding horizon, or do anytime planning. Consequently, we feel that the powerful representation of the graphs of convex sets is underutilized because of the choice of the solving strategy.

We see the aforementioned pitfalls as opportunities and this brings us to INSAT [22, 23, 26] which is a general framework that combines the best of graph search and trajectory optimization. It does so by establishing a symbiotic relationship between the discrete graph that chooses the optimizations to run and the trajectory optimization that decides which part of the discrete graph to explore. Given the sparsity of the GCS graph with respect to the dimension of the planning space and the fact that finding a trajectory via a unary tree of convex sets can be posed as a convex program, INSAT is naturally well-positioned to be a solver for planning on the graphs of convex sets. In this paper, we strongly advocate for using the implicit graph search-based technique INSAT to accelerate planning on the graphs of convex sets. We substantiate our claim by theoretically and experimentally showing that INSATxGCS (IxG) and IxG* are a superior alternative in all our test applications compared to GCS for synthesizing a smooth trajectory. We prove that IxG* has stronger guarantees than GCS on completeness and optimality. IxG*'s stronger completeness enables finding solutions for challenging initial conditions (high velocities etc) and the stronger optimality provides the ability to prespecify the suboptimality bound of the solution as opposed to finding it after the fact in GCS. We provide ample experimental evidence across different applications to show that IxG finds lower-cost solutions orders of magnitude faster than GCS.

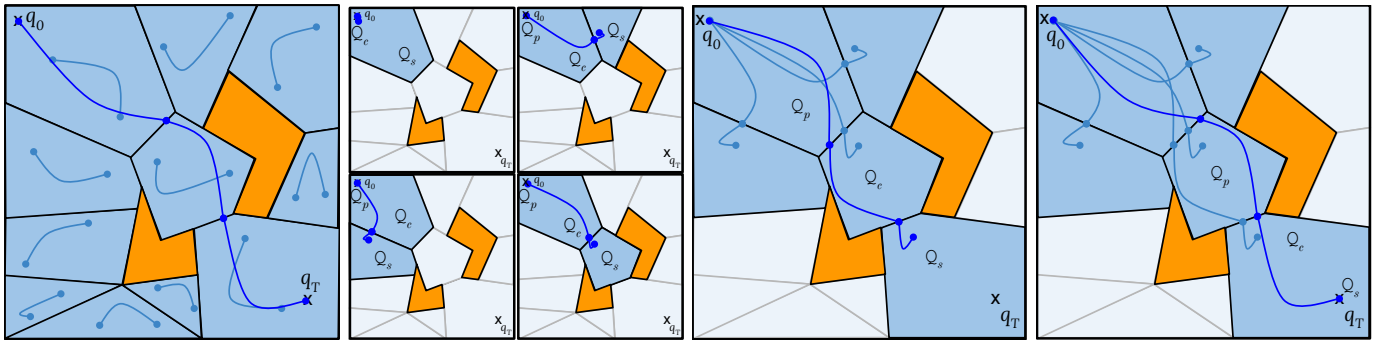
The key idea in this work is based on the observation that depending on the planning query, we need not optimize over the entire graph of convex sets. To materialize this, we interleave searching on the GCS graph and running optimizations only on the fraction of the graph explored by the search. Due to the nature of this optimization in the explored partial graph [2] and the systematic exploration by the search we are able to provide stronger guarantees on completeness and optimality. We will now begin with a brief discussion of the relevant prior work, followed by our proposed method with its theoretical properties, and present the experimental results.

II. Related Work

Collision-free, kinodynamic planning has been a subject of extensive research, driven by the need for robots and autonomous systems to navigate complex environments while considering their dynamic and curvature constraints. This collision avoidance along with dynamic feasibility is typically achieved using one of the four different schemes below

- **Search-based** kinodynamic planning involves precomputing a set of short, dynamically feasible (smooth) trajectories that capture the robot's capabilities called motion primitives. Then search-based planning algorithms like A* and its variants [16] can be used to plan over a set of precomputed motion primitives. These search-based methods provide strong guarantees on optimality and completeness w.r.t the chosen motion primitives. However, the choice and calculation of these motion primitives that prove efficient can be challenging, particularly in high-dimensional systems.

[2]Note that the optimization here can be purely convex as opposed to it being a MICP or a convex relaxation in the batch formulation



(a) Joint GCS optimization (b) Expansion of start state (c) Reusing optimized trajectories (d) INSATxGCS solves faster

Fig. 3: Graphical illustration of INSATxGCS in contrast to GCS. Fig. 3a shows the joint batch optimization performed by GCS over the entire graph. The orange regions denote the obstacles and the blue regions denote the free space decomposed into a graph of convex sets. The final trajectory is shown as a dark blue curve and the optimization over each convex set of the graph is shown as light blue curves. Fig 3b-3d shows the various steps of INSATxGCS incrementally exploring the convex sets and evaluating the edges only as needed. The convex sets in dark blue denote the explored graph and the light ones denote the unvisited convex sets. The trajectories in dark blue are the incremental optimizations that reuse the light blue solutions from previous expansions for computing trajectories to the successor states. INSATxGCS finds the solution trajectory to the goal without even visiting several convex sets of the graph (Fig. 3d).

- **Sampling-based** kinodynamic methods adapt and extend classic approaches such as Probabilistic Roadmaps (PRMs) [12] and Rapidly-exploring Random Trees (RRTs) [15] to handle dynamic systems [14]. This is achieved using dynamically feasible rollouts with random control inputs, solutions of boundary value problems within the *extend* operation, or using geometric primitives that satisfy curvature. There are probabilistically complete and asymptotically optimal variants [10, 13], however, empirical convergence might be tricky.
- **Optimization-based** planning methods can generate high-quality trajectories that are dynamically feasible and do not suffer from the curse of dimensionality in search-based methods. They formulate the motion planning problem as an optimization problem [3] with cost functions defined for trajectory length, time, or energy consumption. After transcribing into a finite-dimensional optimization, these methods rely on the gradients of the cost function and dynamics and employ numerical optimization algorithms to find locally optimal solutions [33, 34, 36]. However, except for a small subset of systems (such as linear or flat systems), for most nonlinear systems these methods lack guarantees on completeness, optimality or convergence. In addition to poor convergence and lack of guarantees, dealing with obstacle constraints in these methods are by far the most challenging compared to other options. To the best of our knowledge, GCS [18, 20] is the first method to turn the optimization among obstacles into a convex program.
- **Hybrid** planning methods combine two or all of the above-mentioned schemes. Search and sampling methods are combined in [8, 32, 17], sampling and optimization methods are combined in [4, 11, 1], search, sampling and optimization methods are combined in [28]. INSAT combined search and optimization methods and demonstrated its capability in several complex dynamical systems [21, 25, 26, 27, 24]. Inspired by INSAT, the approach presented in this paper belongs to this category and combines implicit graph search (Sec. IV-A2) and convex trajectory optimization.

III. Problem Statement

The basic problem statement is the same as motion planning on GCS [20]. We only present a better way to solve this formulation. So for consistency, we borrow the abstract problem formulation from [20]. Consider a set of convex sets $\mathcal{Q} = \{\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_n\} \subset \mathbb{R}^d$ that capture the free and safe planning space for the robot to navigate (Fig. 2-top). These convex sets can be computed in the Cartesian space [6] or in the configuration space for manipulators using [2, 29]. The adjacency relationship between these convex sets is represented as a graph $G_{\mathcal{Q}} = (V_{\mathcal{Q}}, E_{\mathcal{Q}})$ (Fig. 2-bottom). Given these convex sets $\mathcal{Q}$, the goal of the planning algorithm is to find a trajectory $q : [0, T] \to \mathcal{Q}$ that obeys the following optimization

$$\text{minimize } aL(q) + bT \tag{1a}$$
$$\text{subject to } q(t) \in \mathcal{Q}_1 \cup \ldots \cup \mathcal{Q}_n, \forall t \in [0, T] \tag{1b}$$
$$\dot{q}(t) \in \mathcal{D}, \quad \forall t \in [0, T] \tag{1c}$$
$$q(0) = q_0, q(T) = q_T \tag{1d}$$
$$\dot{q}(0) = \dot{q}_0, \dot{q}(T) = \dot{q}_T \tag{1e}$$

where T and $L(q)$ are the duration and arc length of the trajectory $\int_0^T \|\dot{q}(t)\|_2 dt$, $a, b \geq 0$ are the weights corresponding to the relative importance of these terms (Eq. 1a). The constraints require that the trajectory lie entirely within the union of the decomposed convex sets (Eq. 1b), respecting the

velocity limits (Eq. 1c) given by the convex set $\mathcal{D}$ at all times t, and satisfy the boundary conditions (Eq. 1d, 1e) provided by the start state q_0, goal state q_T and their velocities $\dot{q}_0, \dot{q}_T$.

IV. Background

A. Preliminaries

1) Prescribed Graph: A graph $G = (V, E)$ is *prescribed* if the values of vertices and edge weights are prespecified. This is the most common version of a graph. We explicitly introduce this terminology to distinguish it from a graph of convex sets where the vertices are convex sets and edge lengths are convex functions of continuous variables representing the position of the vertices [18]. The combinatorial problem of finding the shortest path on a prescribed graph can be formulated as a simple linear program (LP) [5].

2) Implicit vs Explicit Graph Search: Implicit graph search is a technique that interleaves the construction of the graph with the exploration of space. The performance of graph search algorithms is heavily dominated by the use of appropriate data structures. In many robotics applications, the memory requirement to store the entire graph of the state space explicitly is prohibitive and sometimes even impossible. These problems are solved tractably by interleaving the construction of the graph and searching over it. In the case of graphs of convex sets, although the construction is performed offline and the entire graph is explicitly stored, the trajectories via them are optimized during the search, thereby making the search itself implicit.

3) Lower Bound Graph (LBG): LBG is a lightweight surrogate graph to the underlying graph of convex sets that can be constructed offline using problem-specific parameters. The purpose of LBG is to provide a provable underestimate on the optimal cost between any two states in the graph of convex sets. Once the start and goal states are provided, we run a cheap backward Dijkstra search from the goal state on LBG to generate an admissible heuristic that accelerates IxG and IxG*. The details for the construction of LBG are given in Sec. V-D and its usage within IxG and IxG* is provided in Sec. V-B and V-C.

4) Constructing Graph of Convex Sets: Given the environment, the vertices of the graph of convex sets are constructed by growing convex regions in the free planning space. These regions decompose the planning space and the methods used to grow these regions depend on the problem at hand. They can range from simple axis-aligned rectangles for maze-like environments 6 to decomposing 3D obstacle-free spaces into convex polytopic and ellipsoidal regions [6] and complex configuration spaces into convex hyper-polyhedrons and hyper-ellipsoids [2, 29]. There is also a recent extension to using the clique cover of visibility graph for constructing minimal convex sets with high coverage [37]. In the context of collision-free motion planning, an edge between two regions is considered if they overlap.

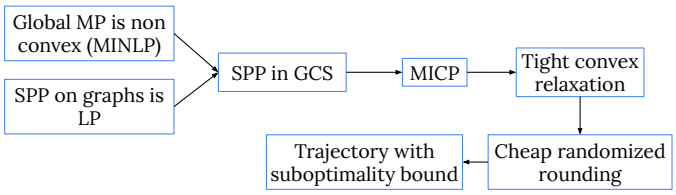


Fig. 4: Working principle of GCS [20].

B. Motion Planning on GCS using Convex Optimization

We provide a brief background on the first work that introduced shortest path planning (SPP) on GCS [20, 18] using the flowchart in Fig. 4. The global motion planning problem of navigating around obstacles is nonconvex. This is because the free planning space that constitutes the domain of the motion planning search/optimization becomes nonconvex even with the presence of one obstacle. Consequently, one needs to solve a nonlinear program (NLP) such as Eq. 1 to solve the motion planning problem. More specifically, since the trajectory $q(t)$ could lie on a finite subset of convex sets $\mathcal{Q}_1, \ldots, \mathcal{Q}_n$, the criteria to identify that subset can be transcribed as an integer constraint over the set in Eq. 1b. Precisely stated, this makes the transcription of optimization in Eq. 1 into a mixed integer nonlinear program (MINLP). The approach in [18] generalizes the idea of using LP for SPP in prescribed graphs to SPP in GCS using a mixed integer convex program (MICP).

Planning on GCS is formulated as MICP where the mixed integer part optimizes for the set of overlapping convex sets to traverse through and the convex optimization computes the trajectory via the overlapping convex sets. In other words, from a graph search perspective, the integer program assigns zeros and ones to edges to decide the optimal edges connecting from start to goal. In GCS, both these subproblems are simultaneously solved in one single optimization. Finding a solution for MICP requires an exact branch and bound algorithm and can be inefficient to solve in practice. GCS proposes a convex relaxation on the edge indicator variables of this MICP and demonstrates that this relaxation is very tight in practice. Once the relaxed values on the edges for the edge variables and the parameters of the choice of trajectory representation within each convex set are determined, a simple depth-first search rounding is performed to extract the final trajectory.

C. INSAT: INterleaved Search And Trajectory Optimization

Grid search is an incredibly powerful tool for planning in state spaces with low dimensions. However, its efficacy diminishes as the dimensionality of the problem grows. In such cases, practitioners often opt for different planning approaches, such as sampling-based planning [12, 15, 14, 13] and optimization-based planning [3, 33, 36, 34]. These alternatives, while valuable, each come with their own set of trade-offs. Sampling-based planning only provides probabilistic completeness; optimization-based planning demands substantial computational resources and is susceptible to getting trapped in local minima.

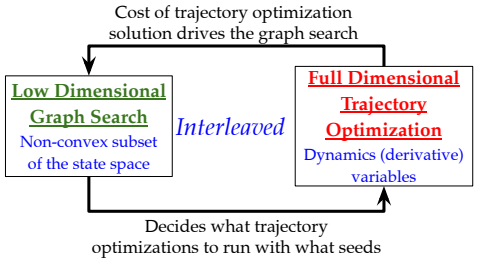


Fig. 5: Working principle of INSAT

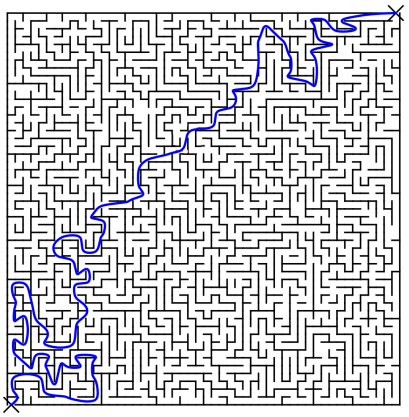


Fig. 6: GCS optimization for planning in this maze (figure borrowed from [20]) has 1,256,157 constraints regardless of the positions of q_0 and q_T.

The INSAT framework provides a mechanism for interleaving discrete search and continuous optimization that aims to address these challenges effectively (Fig. 5). The key idea behind INSAT is (a) to identify a low-dimensional manifold, (b) perform a search over a grid-based or sampling-based graph that represents this manifold, (c) while searching the graph, utilize high-dimensional trajectory optimization to compute the cost of partial solutions found by the search. As a result, the search over the lower-dimensional graph decides what trajectory optimizations to run and with what seeds, while the cost of solution from the trajectory optimization drives the search in the lower-dimensional graph until a feasible high-dimensional trajectory from start to goal is found. The dynamic programming in the low-dimensional search enables warm-starting trajectory optimizations using highly informative initial guesses.

V. Motion Planning on GCS using INSAT

Before we dive into our proposed methods, we will first describe in detail why the GCS representation is underutilized because of the choice of the solution strategy. Following that, we will present our approach of using INSAT as a solver for trajectory optimization on graphs of convex sets leading to two algorithms, IxG and IxG*.

A. GCS Representation is Underutilized

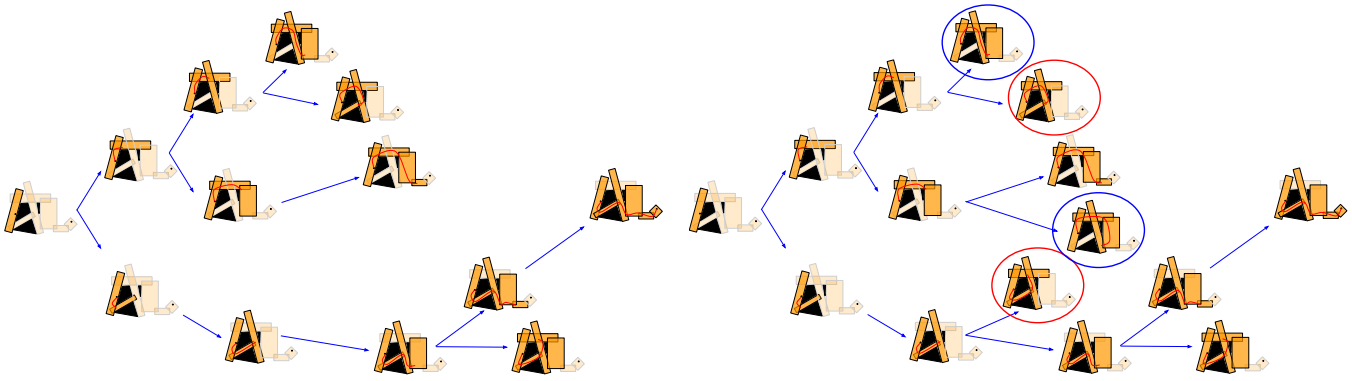
The compact joint optimization is elegant both in terms of the formulation and the result it generates. However, it falls short on four critical fronts namely,

1) As already mentioned, the optimization solves a huge problem even for a trivial planning query which is typically undesirable (Fig. 6).
2) When planning for dynamical systems with challenging initial conditions (with high velocities etc.), to guarantee completeness, the system might have to revisit a state and have a self-intersection in the configuration space (see Sec. V-E1). In the case of GCS, the states are convex sets which could be large subspaces in some instances. The current formulation of GCS does not permit trajectories that require revisiting the same convex set to reach the goal. A trivial modification of maintaining multiple copies of each convex set will make the already big GCS optimization more expensive.
3) The suboptimality bound provided by the GCS method is only computed as an after-the-fact statistic. In other words, GCS does not provide a way to enforce a degree of suboptimality as part of the optimization. This option can be very useful in trading off the solution quality with the planning time in highly complex scenarios.
4) Though the convex decomposition of the planning space is represented using a graph data structure, because of the choice of solution strategy, the GCS method cannot leverage any of the graph search techniques such as anytime or incremental planning, planning with homotopy constraints, search parallelization using distributed hardware etc.

The method presented in this paper precisely addresses these limitations of GCS while having dramatically higher runtime performance.

B. INSATxGCS (IxG)

We now explain IxG in detail using its pseudocode in Alg. 1 and visual illustrations in Fig. 3, 8 and 7. As the relationship between the convex sets is represented using a graph data structure, a graph search algorithm is naturally suited to search over it for trajectory synthesis. Before the search begins, we construct the lower bound graph (LBG) G_{lb} as explained below in Sec. V-D. The algorithm takes as input the start and the goal states q_0 and q_T, the graph of convex sets $G_{\mathcal{Q}}$ and the LBG G_{lb}. It begins by assigning a convex set to q_0 and q_T (Alg. 1: line 13) and updating the LBG to reflect the current planner query with q_0 and q_T (Alg. 1: line 14-15, explained in detail in Sec. V-D). Once the LBG is updated, we run an extremely fast backward Dijkstra search starting from the goal state q_T on the LBG to obtain a provable lower bound from every node to $\mathcal{Q}_T$ in $G_{\mathcal{Q}}$. This lower bound by definition is an under-estimate on the optimal cost-to-go form any $\mathcal{Q}_i$ and can be used as a heuristic to expedite the search [9]. As any informed graph search algorithm, IxG maintains a cost-to-come $g(\mathcal{Q}_i)$ over every convex set in $G_{\mathcal{Q}}$ and initializes $g(Q_0) = 0$. Similar to a best first search like wA* [30], IxG maintains a priority queue called OPEN over a list of convex sets to be expanded.



(a) IxG search progression. (b) IxG* search progression.

Fig. 7: Comparing IxG and IxG* on the same running example introduced in Fig. 2. In IxG graph, the frontier consists of a successor region at most once. But in IxG* graph the same successor region via a different path is present multiple times shown using the same colored circles. This is because IxG* allows re-opening, re-expanding and revisiting states.

Algorithm 1 INSATxGCS (IxG)

1: **procedure** KEY($\mathcal{Q}$) ▷ Computes priority value
2: **return** $g(\mathcal{Q}) + \epsilon * l(\mathcal{Q})$ ▷ Alg. 3
3: **procedure** UPDATELBG($G_{\mathcal{Q}}, G_{lb}, q_i$)
4: $\mathcal{Q}' = \{\mathcal{Q}_i \in V_{\mathcal{Q}} \mid q_i \in \mathcal{Q}_i\}$ ▷ $G_{\mathcal{Q}} = (V_{\mathcal{Q}}, E_{\mathcal{Q}})$
5: **for** $\mathcal{Q}_i \in \mathcal{Q}'$ **do**
6: **for** $v_{lb} \in G_{lb}$ **do**
7: **if** $v_{lb} \in \mathcal{Q}_i$ **then**
8: G_{lb}.AddVertex(q_i) ▷ See Fig. 9, Sec. V-D
9: G_{lb}.AddEdge((q_i, v_{lb})) ▷ See Fig. 9, Sec. V-D
10: G_{lb}.AddEdge((v_{lb}, q_i)) ▷ See Fig. 9, Sec. V-D
11: **return** G_{lb}
12: **procedure** MAIN($q_0, q_T, G_{\mathcal{Q}}, G_{lb}$)
13: $\mathcal{Q}_0 = q_0; \mathcal{Q}_T = q_T$ ▷ Convex sets for q_0 and q_T
14: G_{lb} = UPDATELBG($G_{\mathcal{Q}}, G_{lb}, q_0$) ▷ See Fig. 9
15: G_{lb} = UPDATELBG($G_{\mathcal{Q}}, G_{lb}, q_T$) ▷ See Fig. 9
16: $l(\mathcal{Q})$ = DIJKSTRA(G_{lb}, q_T) ▷ LB search given q_T
17: $\forall \mathcal{Q}_i \in V_{\mathcal{Q}}, g(\mathcal{Q}_i) = \infty$; $g(\mathcal{Q}_0) = 0$ ▷ $G_{\mathcal{Q}} = (V_{\mathcal{Q}}, E_{\mathcal{Q}})$
18: Insert $\mathcal{Q}_0$ in OPEN with KEY($\mathcal{Q}_0$)
19: **while** KEY($\mathcal{Q}_T$) $\leq \infty$ **do**
20: $\mathcal{Q}_c$ = OPEN.pop()
21: $\mathcal{Q}_p$ = Predecessor($\mathcal{Q}_c$)
22: **for** $\mathcal{Q}_s \in$ Successors($\mathcal{Q}_c$) **do**
23: **if** $\mathcal{Q}_s \notin$ CLOSED **then**
24: $\mathcal{Q}^{0\ldots c}$ = Ancestors($\mathcal{Q}_s$)
25: $\mathcal{Q}^{0\ldots s} = (\mathcal{Q}^{0\ldots c}, \mathcal{Q}_s)$
26: $q_{pcs}(t)$ = LBGLOOKUP($\mathcal{Q}_p, \mathcal{Q}_c, \mathcal{Q}_s$) ▷ Sec. V-D
27: $q_{0c}(t) = \mathcal{Q}_c$.trajectory() ▷ From recursion
28: $q_{0s}(t)$ = OPTIMIZE($\mathcal{Q}^{0\ldots s}, q_{0c}(t), q_{pcs}(t)$) ▷ Eq. 2
29: **if** $q_{0s}(t)$.isValid() **then**
30: **if** $c(q_{0s}(t)) < g(q_{0s}(t))$ **then** ▷ Eq. 2a
31: $\mathcal{Q}_s$.trajectory = $q_{0s}(t)$
32: Insert/Update $\mathcal{Q}_s$ in OPEN with KEY($\mathcal{Q}_s$)
33: **return** $\mathcal{Q}_T$.trajectory()

Inside the search loop that runs until the goal state is expanded (Alg. 1: line 19), IxG picks the lowest cost node $\mathcal{Q}_c$ for expansion (Alg. 1, line 20) per the priority value (Alg. 1: line 1). The priority value is a sum of cost-to-come $g(\mathcal{Q})$ and weighted cost-to-go $\epsilon l(\mathcal{Q})$. The term ϵ is an inflation on the admissible heuristic (underestimate) that trades off planning speed to solution quality [30]. It is important to note that the search also keeps track of the expanded nodes using CLOSED list (Alg. 1, line 23) to prevent re-expansion. As IxG is derived by applying INSAT to plan on GCS, for every successor $\mathcal{Q}_s$ generated from expanding $\mathcal{Q}_c$, we consider the set of ancestors $\mathcal{Q}^{0\ldots s}$ and optimize a trajectory through them via a two-step process (Alg. 1, lines 22-28). Let $\mathcal{Q}_p$ be the predecessor of $\mathcal{Q}_c$ (see Fig. 3). First, an incremental trajectory $q_{pcs}(t)$ that connects any point in $\mathcal{Q}_p$ to any point $\mathcal{Q}_s$ via $\mathcal{Q}_c$ is computed. We will see in Sec. V-D how this step can fast-forwarded by a simple lookup. Then, the full trajectory from the start state $q_{0s}(t)$ is obtained by warm-starting an optimization over $\mathcal{Q}^{0\ldots s}$ with $q_{0c}(t)$ and $q_{pcs}(t)$ (Sec. V-B1, Eq. 2). Note that the dynamic programming nature of the algorithm will guarantee the existence of $q_{0c}(t)$, as otherwise $\mathcal{Q}_c$ would not have been added to OPEN and therefore never expanded. Finally, the optimized trajectory $q_{0s}(t)$ is checked for validity and added/updated in the OPEN list to be expanded in the future. When the goal convex region $\mathcal{Q}_T$ is generated as a successor, if a valid trajectory to it $q_{0T}(t)$ is computed, then the condition in line 19 fails. The search then exits the loop and returns $q_{0T}(t)$.

1) Optimization over a Sequence of Convex Sets: In line 28 of Alg. 1, we solve an optimization given a sequence of convex sets, namely, the one obtained from the set of ancestors. This step is a crucial difference between the GCS trajectory optimization and IxG and perhaps the single most important reason for the efficiency of IxG. IxG decouples the joint GCS optimization of choosing the safe set of convex regions to traverse and designing robot trajectories within each region into searching over the safe set of convex regions,

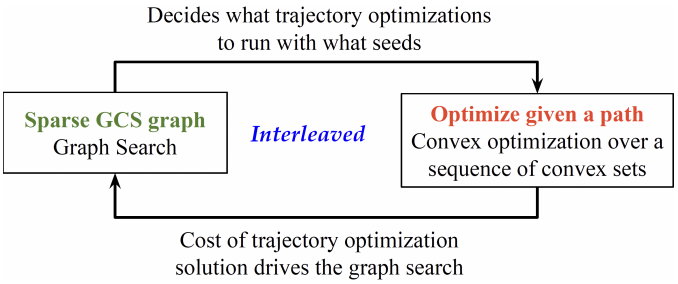


Fig. 8: Working principle of IxG, IxG*

only optimizing trajectories over these partial paths from this search and guiding the search using the output of the optimized trajectory. As a result, the MICP in GCS trajectory optimization is split into a graph search that explores the convex regions systematically and a convex optimization over a prespecified sequence of convex sets.

Consider a prespecified path given by a sequence of convex regions $\mathcal{Q}^{1\dots K} = (\mathcal{Q}_1, \mathcal{Q}_2, \dots, \mathcal{Q}_K)$ in the graph $G_\mathcal{Q}$ such that $(\mathcal{Q}_k, \mathcal{Q}_{k+1}) \in E_\mathcal{Q}$ where $1 < k \leq K \leq n$. The piecewise trajectory through the path $\mathcal{Q}^{1\dots K}$ made of a trajectory segment through each region $q_k(t)$ can be found by solving the below convex optimization. This is the operation carried out in line 28 of Alg. 1 and line 21 of Alg. 2.

$$\min\ aL(q) + b\sum_{k=1}^{K} T_k \tag{2a}$$

$$\text{s.t. } q_k(t_k) \in \mathcal{Q}_k \qquad \forall t_k \in [0, T_k], k \in 1, \dots, K \tag{2b}$$

$$\dot{q_k}(t_k) \in \mathcal{D} \qquad \forall t_k \in [0, T_k], k \in 1, \dots, K \tag{2c}$$

$$q_{k-1}^{(j)}(T_k) = q_k^{(j)}(0) \quad \forall t_k \in [0, T_k], k \in 2, \dots, K \tag{2d}$$

where Eq. 2b requires every trajectory piece $q_k(t_k)$ to lie in their corresponding convex set $\mathcal{Q}_k$, Eq. 2c is similar to Eq. 1c and Eq. 2d captures overall continuity and smoothness using a boundary condition for each segment of the trajectory and its derivative such that the endpoint of a segment matches the starting point of the subsequent segment. In Alg. 1 and Alg. 2 the start and goal states within the start and goal convex sets are represented as convex sets too (the start/goal state is a point in the ambient space, and is trivially a convex set). So the start state has an outgoing edge to the convex set it belongs to and the goal state has an incoming edge from the convex set it belongs to. Hence under this representation, the explicit boundary conditions similar to Eq. 1d and Eq. 1e can be removed in Eq. 2.

C. IxG*

Before we describe the details of the optimal version of IxG called IxG*, we will briefly explain why IxG is not an optimal algorithm. An important caveat in considering the set of ancestors to optimize for a trajectory edge to every successor during expansion is that it breaks the Markov property of heuristic search. The Markov property in graph search requires that the cost of the successor depends only on the current state and not on the history of states leading up to it. Heuristic

Algorithm 2 IxG*

1: **procedure** KEY($\mathcal{Q}^{a\dots b}$)
2: **return** $g(\mathcal{Q}^{a\dots b}) + \epsilon * l(\mathcal{Q}_b)$ ▷ Get $\mathcal{Q}_b$ from $\mathcal{Q}^{a\dots b}$
3: **procedure** UPDATELBG($\mathcal{Q}, G_{lb}, q_i$)
4: Same as Alg. 1
5: **procedure** MAIN($q_0, q_T, \mathcal{Q}, G_{lb}$)
6: Same as lines 13-16 of Alg. 1
7: $q_{0T}(t)$ =IxG($q_0, q_T, \mathcal{Q}, G_{lb}$) ▷ Use Alg. 1 for UB
8: $u = \epsilon * c(q_{0T}(t))$ ▷ UB on bounded suboptimal cost
9: $\forall \mathcal{Q}^{0\dots T}, g(\mathcal{Q}^{0\dots T}) = \infty$; $g(\mathcal{Q}^0) = 0$
10: Insert $\mathcal{Q}^0$ in OPEN with KEY($\mathcal{Q}^0$)
11: **while** KEY($\mathcal{Q}^{0\dots T}$) $\leq$ OPEN.min() **do**
12: $\mathcal{Q}^{0\dots c}$ = OPEN.pop()
13: $\mathcal{Q}_p$ = Predecessor($\mathcal{Q}_c$) ▷ Get $\mathcal{Q}_c$ from $\mathcal{Q}^{0\dots c}$
14: **for** $\mathcal{Q}_s \in$ Successors($\mathcal{Q}_c$) **do** ▷ Get $\mathcal{Q}_c$ from $\mathcal{Q}^{0\dots c}$
15: **if** Allow Cycles **or** $\mathcal{Q}_s \notin \mathcal{Q}^{0\dots c}$ **then**
16: $\mathcal{Q}^{0\dots s} = (\mathcal{Q}^{0\dots c}, \mathcal{Q}_s)$
17: **else if** $\mathcal{Q}_s \in \mathcal{Q}^{0\dots c}$ **then**
18: **continue**
19: $q_{pcs}(t)$ = LBGLOOKUP($\mathcal{Q}_p, \mathcal{Q}_c, \mathcal{Q}_s$) ▷ Sec. V-D
20: $q_{0c}(t) = \mathcal{Q}^{0\dots c}$.trajectory() ▷ From recursion
21: $q_{0s}(t)$ = OPTIMIZE($\mathcal{Q}^{0\dots s}, q_{0c}(t), q_{pcs}(t)$) ▷ Eq. 2
22: **if** $q_{0s}(t)$.isValid() **then**
23: **if** $c(q_{0s}(t)) + \epsilon * l(\mathcal{Q}_s) > \epsilon * u$ **then**
24: **continue** ▷ Prune the path $\mathcal{Q}^{0\dots s}$
25: **else**
26: $\mathcal{Q}^{0\dots s}$.trajectory = $q_{0s}(t)$
27: Insert/Update $\mathcal{Q}^{0\dots s}$ in OPEN with KEY($\mathcal{Q}^{0\dots s}$)
28: **return** $\mathcal{Q}^{0\dots T}$.trajectory()

search methods leverage this property to introduce CLOSED list and guarantee optimality under admissible heuristics without re-opening and re-expanding states. As IxG violates the Markov property, it is neither optimal nor bounded suboptimal. Consequently, we need to allow the re-expansion of the convex regions in the search and essentially perform a tree search. Just by allowing the re-expansion of the states (with and without allowing cycles) and thereby searching over all possible paths for every single state, IxG can be made provably optimal and bounded suboptimal. However, this will blow up the number of expansions and make the algorithm intractable. To alleviate this, we introduce a pruning mechanism (Alg. 2, line 23) using a lower bound on the cost-to-go from every state (we use LBG computed in Alg. 3) and an upper bound on the optimal cost (Alg. 2, line 7-8).

IxG* algorithm presented using pseudocode in Alg. 2 inherits most of its procedure from IxG (Alg. 1). The key differences are (i) the search is carried over the paths instead of convex states (see line 12 in Alg. 2) (ii) the CLOSED list is omitted in favor of allowing re-opening, re-expanding and revisiting (allow cycles in line 15) convex regions and (iii) the pruning criteria employed in line 23. The pruning step checks if the sum of the cost-to-come via a particular path and the cost-to-go obtained from LBG is greater than the maximum allowed cost computed as upper bound at the beginning of

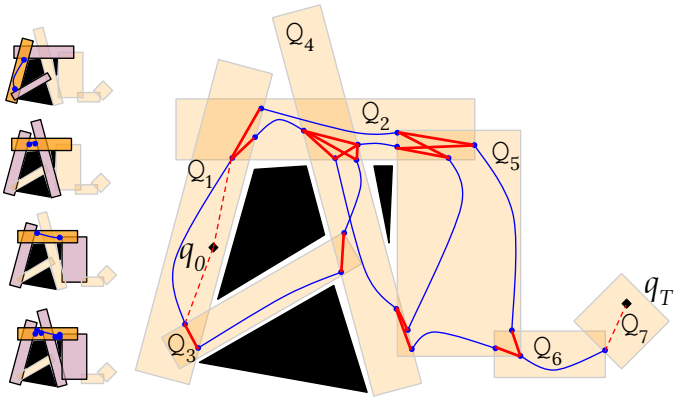


Fig. 9: The lower bound graph (LBG) whose vertices are shown as blue dots and edges are shown using blue curves and red lines. The blue curves are found by solving a convex program over every convex set triplet connected by edges (shown in the left column with the triplet highlighted as purple-orange-purple). The red edges are optimized with simpler constraints or can be set to zero cost to satisfy the lower bound trivially. See Alg. 3 for construction of LBG.

the search. If this condition is satisfied, then such a path will never lead to an optimal or ϵ-suboptimal solution. As we are searching different paths of convex sets leading to the same convex region, the OPEN list contains a list of paths.

D. Lower Bound Graph (LBG)

The lower bound graph (LBG) is the core accelerator behind the efficient performance of IxG*. Given a choice of motion generation parameters such as the order, minimum derivative of continuity, etc in the case of B-splines trajectories, the LBG is a surrogate *prescribed graph* computed over $G_{\mathcal{Q}}$ using Alg. 3. The edges of the LBG are formed by optimizing over a 3-sequence (triplet) of convex sets in $G_{\mathcal{Q}}$ (Alg. 3, line 9-10). Each triplet constitutes a node in $G_{\mathcal{Q}}$ along with its incoming and outgoing convex regions (Alg. 3, lines 4-8). These edges are then added to the LBG graph G_{lb}. The boundary points of the optimized edge trajectories form V_{lb}. Following the edges generated from triplet optimization (see blue curves in Fig. 9), the graph is made into a single connected component by joining the terminals of optimized trajectories in the overlapping subset of convex regions with zero cost edges (Alg. 3, lines 15-18) or edges with lower order and continuity (to certify provable lower bound). This is shown as red lines in Fig. 9. Finally, the Alg. 3 returns the constructed G_{lb} with a single component.

The LBG graph is used for two purposes in IxG/IxG*. Using an optimal search such as Dijkstra over G_{lb} gives a provable lower bound on the optimal cost to traverse between any two convex regions in $G_{\mathcal{Q}}$. Given a planning query, q_0 and q_T, the LBG is used for computing the admissible heuristic from any state to q_T and verifying the pruning criteria in IxG*. Moreover, since the LBG is formed by optimizing over the set of possible triplets in $G_{\mathcal{Q}}$, they can be used directly as the solutions of incremental optimization steps in IxG and IxG* (Alg. 1, line 26 and Alg. 2, line 19)

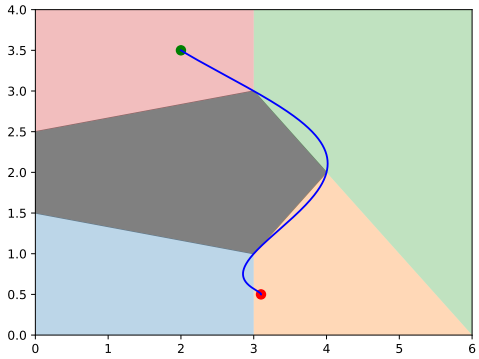


Fig. 10: A simple 2D example with one obstacle (grey) illustrates the need to revisit a convex set to find a smooth trajectory from the start (red dot) to the goal (green dot). The free space is decomposed into four convex sets. In this example, the initial velocity at the start state in the orange convex set is set to a high nonzero value in the direction of the blue convex set to its left. As there are constraints on the smoothness of the trajectory at the convex set boundaries, the only solution is to exit the start orange set, enter the blue set, exit the blue set, and enter the orange set again before arriving at the goal. Finding this trajectory requires re-expanding the orange convex set during search which is not addressed in the original GCS trajectory optimization [20].

E. Theoretical Analysis

1) Stronger Completeness: The stronger completeness discussed in this section is easier to appreciate in the context of dynamical systems. Consider a point robot with a high nonzero velocity starting near the edge of a convex set facing the direction of the neighboring convex set (Fig. 10). To guarantee smoothness, the robot might have to enter the neighboring convex set and revisit the current convex set and this may have to be done several times depending on the environment. The current GCS formulation does not provide a way to handle this scenario and a trivial modification of maintaining multiple copies of each convex set will make the already big GCS optimization further expensive in addition to not guaranteeing completeness [3].

To combat this, the graph search underpinning the IxG* does not require prespecifying the number of times a convex region must be visited to find a solution and guarantee completeness. Instead, as the graph is built implicitly, the number of cycles can be adapted on demand and as per the need (Alg. 2, line 15).

2) Trivial Parallelization of IxG:* Since the graph search component of IxG* searches over paths of convex sets and maintains a separate copy of every way to reach a particular convex set, there is no interdependence between nodes picked for expansion. As a result, depending on the thread budget, any number of nodes can be popped from the OPEN list in line 12 of Alg. 2 for concurrent expansion. The optimality and completeness of the algorithm will not be sacrificed so long as

[3]The exact number of times a region has to be revisited cannot be exactly known ahead of time.

Algorithm 3 Lower Bound Graph (LBG) search

procedure LBG($G_{\mathcal{Q}}$)
  $G_{lb} = (V_{lb}, E_{lb})$; $V_{lb} = E_{lb} = \emptyset$
  for $\mathcal{Q}_c \in V_{\mathcal{Q}}$ **do** ▷ $G_{\mathcal{Q}} = (V_{\mathcal{Q}}, E_{\mathcal{Q}})$
    for $\mathcal{Q}_p \in$ Neighbors($\mathcal{Q}_c$) **do**
      for $\mathcal{Q}_s \in$ Neighbors($\mathcal{Q}_c$) **do**
        if $\mathcal{Q}_p \neq \mathcal{Q}_c$ **then**
          $\mathcal{Q}^{\dagger} = (\mathcal{Q}_p, \mathcal{Q}_c, \mathcal{Q}_s)$
          $\mathcal{Q}^{\ddagger} = (\mathcal{Q}_s, \mathcal{Q}_c, \mathcal{Q}_p)$
          $q_{pcs}(t)$ = Optimize($\mathcal{Q}^{\dagger}$) ▷ Eq. 2
          $q_{scp}(t)$ = Optimize($\mathcal{Q}^{\ddagger}$) ▷ Eq. 2
          V_{lb}.Add($q_{pcs}(0)$); V_{lb}.Add($q_{pcs}(T)$)
          V_{lb}.Add($q_{scp}(0)$); V_{lb}.Add($q_{scp}(T)$)
          E_{lb}.Add($q_{pcs}(t)$) with cost $c(q_{pcs}(t))$
          E_{lb}.Add($q_{scp}(t)$) with cost $c(q_{scp}(t))$
  for $(\mathcal{Q}_1, \mathcal{Q}_2) \in E_{\mathcal{Q}}$ **do** ▷ $G_{\mathcal{Q}} = (V_{\mathcal{Q}}, E_{\mathcal{Q}})$
    for $q_1, q_2 \in V_{lb}$ such that $q_1 \neq q_2$ **do**
      if $q_1, q_2 \in \mathcal{Q}_1 \cap \mathcal{Q}_2$ **then**
        E_{lb}.Add((q_1, q_2)) with cost 0 ▷ LB edge
  return G_{lb}

the order of expansion and the priority value are maintained.

3) Properties of IxG, IxG and LBG:* Table. I lists the properties of optimality and completeness of IxG and IxG*. Detailed proofs of the properties are provided in Appendix 1 of the supplementary material. As noted before, the optimality and bounded suboptimality properties are stronger than what was provided for the GCS trajectory optimization [20]. This is because IxG* enforces a factor of suboptimality of the solution in Alg. 2 whereas it is only calculated as an after-the-fact statistic in [20]. As explained above, IxG* also satisfies a stricter definition of completeness.

In the same vein, the proofs on the bound on the number of nodes in LBG and guarantees of generating provably admissible heuristic is provided in Appendix 2 of the supplementary material.

	Optimality	Bounded Suboptimality	Completeness
IxG	✗	✗	✓[4]
IxG*	✓	✓	✓

TABLE I: Optimality and completeness of IxG and IxG*.

VI. Experimental Results

We evaluate the empirical performance of IxG and IxG* in simulation against GCS in three different applications: (1) a 2D system in a 50×50 maze environment, (2) UAV in a 3D highly cluttered environment with trees and buildings and (3) an assembly cell with three Motoman HC10DTP arms for assembly. All the methods are implemented in C++ in the backend and sometimes invoked from Python wrappers. Tests ran on a 128-core AMD Ryzen Threadripper Pro with 512GB of memory.

[4]Under the assumption that all the edge constraints can be satisfied

A. 2D Maze

For the 2D maze environment, we utilized the same maze provided in [20] and tested using 50 randomly sampled start-goal pairs. The graph of convex sets is constructed by considering axis-aligned rectangles between the walls of the maze and their overlaps. The goal of the planner is to generate a trajectory with C^2 continuity from start to goal. In this environment, there are 2500 convex sets and 5198 edges in the graph. The results show that IxG* consistently outperforms GCS in terms of planning time. However, with $\epsilon = 6$, the solution cost is slightly larger than that of GCS (Table. II). This is exactly the kind of trade-off between solution quality and planning time that can be obtained by controlling and pre-specifying the suboptimality factor.

	GCS	IxG* ($\epsilon = 6$)
Success Rate (%)	100%	100%
Solution Cost	52.636	52.832
Planning Time (s)	6.728	1.2613
# Optimized Edges	5198	440.94

TABLE II: Various statistics show IxG* outperforming GCS in the 2D maze environment. Note that the higher solution cost of IxG* is because of planning with $\epsilon = 6$.

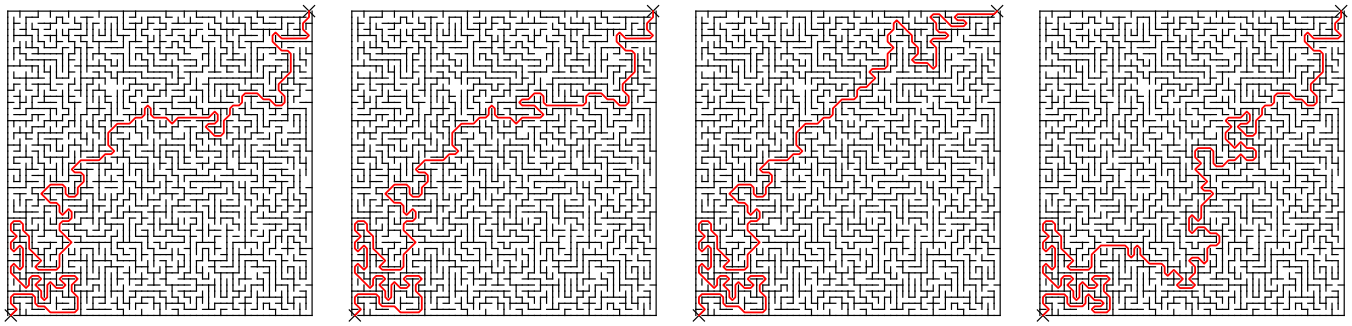
For the same pair of start and goal states, we ran IxG* with different inflations of the admissible LBG heuristic to show how the solution changes as we increase the upper bound on the cost of the optimal solution (see Fig. 11). Fig. 14 shows how the solution quality and the runtime of the planner are impacted by ϵ. As discussed before, allowing re-opening, re-expanding, and revisiting states in IxG* can blow up the complexity of the search. This was controlled by the efficient pruning mechanism using LBG proposed in Sec. V-C. We numerically show the effect of this pruning in Fig. 12. The drastic reduction (see different scales in the color bar) in the number of node re-expansions (top vs bottom) implies the tightness of the lower bounded computed by LBG. The LBG used for pruning is provided in Fig. 13. Finally, we plot the effect of suboptimality factor (heuristic inflation) vs planning time in Fig. 15. As expected, we see that the planning time of IxG* goes down when allowing higher suboptimality of the solution. GCS trajectory optimization takes the same amount of time as there is no way to enforce lower fidelity of solutions.

	GCS	IxG* ($\epsilon = 10$)
Success Rate (%)	100%	100%
Solution Cost	16.055	8.676
Planning Time (s)	112.867	1.85
# Optimized Edges	12346	211.6

TABLE III: Statistics showing IxG* outperforming GCS in the 15m×15m UAV forest environment. Note >60x improvement in planning time.

B. 3D UAV

We conducted experiments in randomly generated villages, complete with trees and buildings. These maps are the same



(a) $\epsilon = 4$ (b) $\epsilon = 8$ (c) $\epsilon = 16$ (d) $\epsilon = 64$

Fig. 11: Continuous trajectory synthesized by IxG* from one end of the maze to another for different inflations of the heuristic. These inflations are the factor of an upper bound on the cost of the optimal solution.

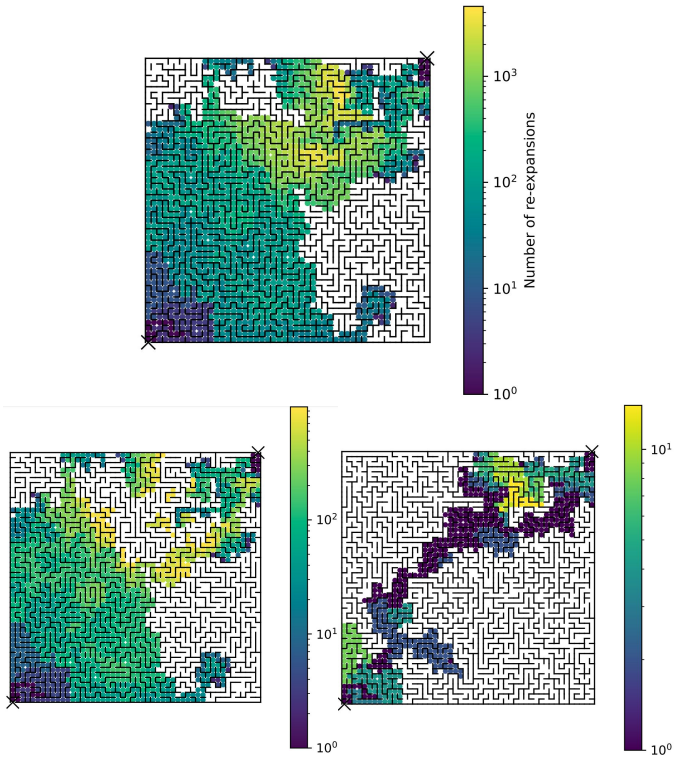
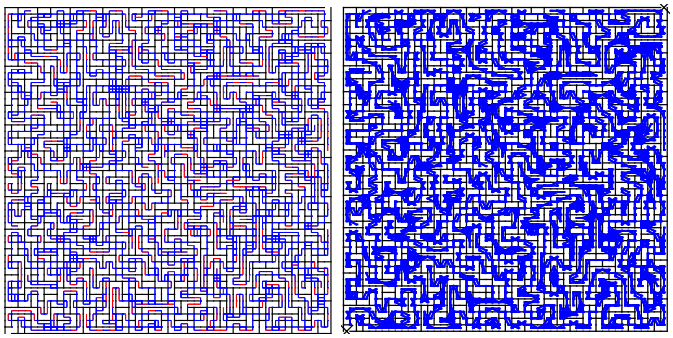


Fig. 12: The effect of pruning using the LBG in IxG* as explained in Sec. V-C, V-D. The color bar (see different scales for each) denotes the number of re-expansions of a convex region to satisfy the suboptimality factor. The figure at the top is without the pruning operation (*i.e.* $u = \infty$ in Alg. 2, line 23) for $\epsilon = 1$. The bottom figures denote the number of re-expansions with pruning for $\epsilon = 1$ (left) and $\epsilon = 6$ (right).

	IxG* ($\epsilon = 15$)
Success Rate (%)	96%
Solution Cost (rad)	16.44
Planning Time (s)	31.624
# Optimized Edges	1832.24

TABLE IV: Results of IxG* for multi-arm manipulation planning. The GCS method was unable to load this problem into memory.



(a) Trajectory is represented as B-splines with order=1. (b) Trajectory is represented as B-splines with order=3.

Fig. 13: Lower bound graph (LBG) computed for the 2D maze scenario. The red lines denote zero cost edges.

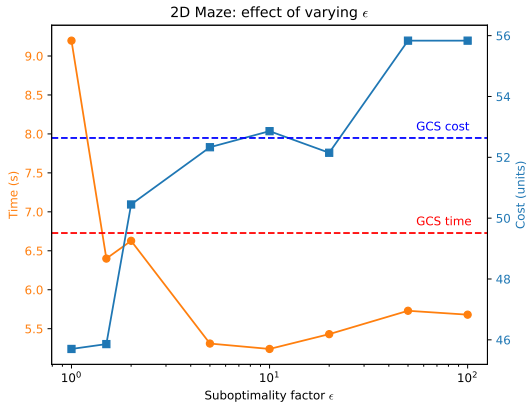


Fig. 14: Effect of varying ϵ on the planning time and the solution cost for 2D maze.

as what is used in [19] and the graph of convex sets is made of axis-aligned 3D boxes and their overlaps. In a 50m × 50m map, our environment comprised over 10,000 convex sets and 140,000 edges. In a 15m × 15m map, we had 900 convex sets and over 12,000 edges. For our experiments, we focused on the 15m × 15m village, where we compared the proposed IxG* against GCS (Fig. 16, Table. III). We refrained from comparing IxG/IxG* with GCS on the 50m × 50m map. This was due

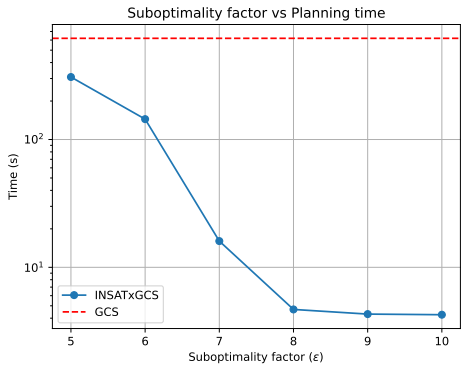


Fig. 15: Decrease in planning time of IxG as we increase the suboptimality factor for UAV planning in 50m×50m map. The GCS planning time is shown using a red dotted line

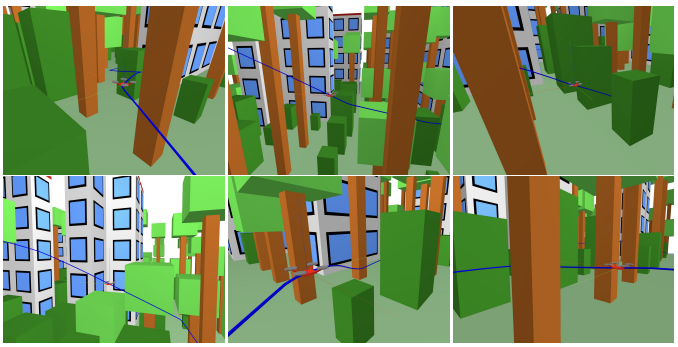


Fig. 16: UAV flying through dense forest in simulation using plans generated with INSATxGCS

to GCS requiring more than 600 seconds to find solutions, whereas INSATxGCS exhibited an average planning time of just 5.119 seconds. We represented our trajectories using B-spline curves of order 5. We also compared against a recently proposed fast method called FPP for this environment [19]. FPP took an average of 2.83s for 15m×15m environment and 5.507s for 50m×50m, thus taking longer than IxG* for both sizes of the map. Nonetheless, it is important to note that [19] is applicable only for the graph of convex sets made of axis-aligned boxes and does not have any guarantees on the optimality of the solution unlike IxG*.

C. Multi-Arm Manipulation

The setup consists of three 6-DoF Motoman HC10DTP robot arms positioned around a C-shaped table (Fig. 1). The 18-dimensional configuration space is decomposed into 2092 regions using task-specific seeds of interest with a 68% coverage. With these seeds, the regions are grown using nonlinear programming as explained in [29] using the implementation in Drake [35] (`IrisInConfigurationSpace()`). The regions had a significant overlap among them leading to 133K edges. For these experiments, we restricted the maximum degree of the graph of convex sets to 8 and pruned significantly overlapping and redundant regions to a total of 67 regions with 2224 edges. We randomly sampled start and goal configurations with a deliberate bias to pick hard combinations that require

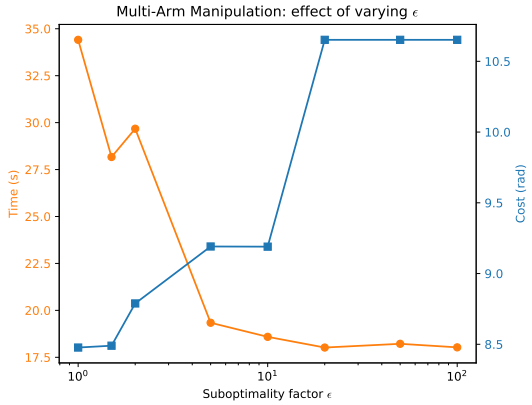


Fig. 17: Effect of varying ϵ on the planning time and the solution cost for the multi-arm manipulation.

intricate movements between the arms while being very close to each other. For this experiment, we could not compare against GCS as it was unable to transcribe the whole graph into costs and constraints and load it entirely into memory. We even tried GCS with random subgraphs of the full graph used for experiments with IxG* and found that the maximum size of the subgraph that GCS solved had 10 vertices (compared to 67 in the original). Even for a subgraph of size 10, GCS took about 163s to solve and is already 5.2x slower than IxG*.

IxG* exhibited promising performance, with an average planning time of 31.62s seconds (Table. IV). Since IxG* enables informing the search with a suboptimality factor, the quality of the solution can be traded off with the speed of the planner. Fig. 17 shows how the solution quality and the runtime of the planner are impacted by ϵ. It can be noted that the planning time decreases and solution quality gets worse with increasing suboptimality factor.

D. Reduction in Complexity

As a consequence of using implicit search to plan over the graph of convex sets, there is a reduction in the complexity of the planning algorithm across several metrics like reduced size of the problem, a significant drop in the largest optimization call, etc. Some of these metrics were already reported in the above Tables. The size of the GCS graph for different domains are given in Table. V along with the giant constant size of the optimization for any query in Table. VI. As noted, IxG and IxG* solves many small optimization problems and almost never solves one of GCS' size. The largest optimization problem invoked by IxG* is given in Table. VII.

	# Convex Sets	# Edges
2D Maze	2500	5198
UAV (15m × 15m)	900	12346
UAV (50m × 50m)	10105	140506
Motoman HC10DTP 3x Arms	2092	12445

TABLE V: Details about the graph of convex sets

E. Implementation Details

An important implementation detail that contributes to the efficiency of IxG and IxG* lies in how the costs and constraints

	# Decision Vars	# Costs	# Constraints
2D Maze	119586	15202	256261
UAV (15m × 15m)	555657	17751	759756
UAV (50m × 50m)	6322851	201141	8642649

TABLE VI: GCS optimization problem size

	2D Maze	UAV (15m × 15m)	UAV (50m × 50m)
Max # Decision Vars	1845	196	1175

TABLE VII: Largest optimization IxG* invoked

are built for the optimization. Though the graph is explored and the optimization is solved implicitly, for a given decomposition of the graph, the symbolic costs and constraints of all the GCS vertices and edges over the entire graph are constructed and cached offline. During runtime, when the search explores a particular path in the GCS graph, the cached symbolic costs and constraints are retrieved at virtually no cost and supplied to the convex solver.

VII. Conclusion

We showed that using INSAT to synthesize trajectories over the graph of convex sets is a better alternative compared to the GCS batch optimization. To that end, we developed two algorithms IxG and IxG*, and discussed trivial extensions to parallelized IxG* without sacrificing any of the theoretical properties. We can plan orders of magnitude faster with better theoretical properties. Graphs of convex sets tackled one of the biggest bottlenecks of collision checking in motion planning by representing the free planning space as a union of convex sets stored as a graph. This representation can make motion planning dramatically simple as every pair of points in a convex set are connected by a straight line and as optimization over convex sets is extremely well studied. However joint optimization over the entire graph is unnecessary in most instances and can limit the capability of the GCS representation. Using a search-based method also brings the advantage of all the planning techniques developed in discrete graph search. For example, there are many replanning and anytime algorithms in graph search that efficiently reuse search efforts in dynamically changing environments or return a quick time-bounded solution while improving the solution quality. We believe that the introduction of a graph search-based framework such as INSAT to explore and prune the convex sets while optimizing the trajectory via them widens the scope and application of GCS.

VIII. Acknowledgements

This work was supported by grants W911NF-21-1-0050 and W911NF-18-2-0218 of the ARL-sponsored A2I2 program. We thank Yorai Shaoul for his help in rendering some of the visualizations presented in this work.

References

[1] Kalyan Vasudev Alwala and Mustafa Mukadam. Joint sampling and trajectory optimization over graphs for online motion planning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4700–4707. IEEE, 2021.

[2] Alexandre Amice, Hongkai Dai, Peter Werner, Annan Zhang, and Russ Tedrake. Finding and optimizing certified, collision-free regions in configuration space for robot manipulators. In *International Workshop on the Algorithmic Foundations of Robotics*, pages 328–348. Springer, 2022.

[3] John T Betts. *Practical methods for optimal control and estimation using nonlinear programming*. SIAM, 2010.

[4] Sanjiban Choudhury, Jonathan D Gammell, Timothy D Barfoot, Siddhartha S Srinivasa, and Sebastian Scherer. Regionally accelerated batch informed trees (rabit*): A framework to integrate local information into optimal path planning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4207–4214. IEEE, 2016.

[5] Sanjoy Dasgupta, CH Papadimitriou, and Umesh Vazirani Algorithms. Mcgraw-hill science. *Engineering/Math*, 2006.

[6] 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.

[7] Moritz Diehl, Hans Georg Bock, Holger Diedam, and P-B Wieber. Fast direct multiple shooting algorithms for optimal robot control. *Fast motions in biomechanics and robotics: optimization and feedback control*, pages 65–93, 2006.

[8] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 3067–3074. IEEE, 2015.

[9] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[10] Kris Hauser and Yilun Zhou. Asymptotically optimal planning by feasible kinodynamic planning in a state–cost space. *IEEE Transactions on Robotics*, 32(6):1431–1443, 2016.

[11] Jay Kamat, Joaquim Ortiz-Haro, Marc Toussaint, Florian T Pokorny, and Andreas Orthey. Bitkomo: Combining sampling and optimization for fast convergence in optimal motion planning. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4492–4497. IEEE, 2022.

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

[13] Tobias Kunz and Mike Stilman. Kinodynamic rrts

with fixed time step and best-input extension are not probabilistically complete. In *Algorithmic Foundations of Robotics XI: Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics*, pages 233–244. Springer, 2015.

[14] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. *Int. J. Robot. Research*, 20(5): 378–400, 2001.

[15] Steven M LaValle et al. Rapidly-exploring random trees: A new tool for path planning. 1998.

[16] Maxim Likhachev and Dave Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research*, 28(8): 933–945, 2009.

[17] Zakary Littlefield and Kostas E Bekris. Efficient and asymptotically optimal kinodynamic motion planning via dominance-informed regions. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9. IEEE, 2018.

[18] Tobia Marcucci, Jack Umenberger, Pablo A Parrilo, and Russ Tedrake. Shortest paths in graphs of convex sets. *arXiv preprint arXiv:2101.11565*, 2021.

[19] Tobia Marcucci, Parth Nobel, Russ Tedrake, and Stephen Boyd. Fast path planning through large collections of safe boxes. *arXiv preprint arXiv:2305.01072*, 2023.

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

[21] Ramkumar Natarajan, Howie Choset, and Maxim Likhachev. Interleaving graph search and trajectory optimization for aggressive quadrotor flight. *IEEE Robotics and Automation Letters*, 6(3):5357–5364, 2021.

[22] Ramkumar Natarajan, Howie Choset, and Maxim Likhachev. Interleaving graph search and trajectory optimization for aggressive quadrotor flight. *IEEE Robotics and Automation Letters*, 6(3):5357–5364, 2021. doi: 10.1109/LRA.2021.3067298.

[23] Ramkumar Natarajan, Garrison L. H. Johnston, Nabil Simaan, Maxim Likhachev, and Howie Choset. Torque-limited manipulation planning through contact by interleaving graph search and trajectory optimization, 2022.

[24] Ramkumar Natarajan, Garrison L Johnston, Nabil Simaan, Maxim Likhachev, and Howie Choset. Long-horizon torque-limited planning through contact using discrete search and continuous optimization. In *IROS 2023 Workshop on Leveraging Models for Contact-Rich Manipulation*, 2023.

[25] Ramkumar Natarajan, Garrison LH Johnston, Nabil Simaan, Maxim Likhachev, and Howie Choset. Torque-limited manipulation planning through contact by interleaving graph search and trajectory optimization. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8148–8154. IEEE, 2023.

[26] Ramkumar Natarajan, Shohin Mukherjee, Howie Choset, and Maxim Likhachev. Pinsat: Parallelized interleaving of graph search and trajectory optimization for kinodynamic motion planning. *arXiv preprint arXiv:2401.08948*, 2024.

[27] Ramkumar Natarajan, Hanlan Yang, Qintong Xie, Yash Oza, Manash Pratim Das, Fahad Islam, Muhammad Suhail Saleem, Howie Choset, and Maxim Likhachev. Preprocessing-based kinodynamic motion planning framework for intercepting projectiles using a robot manipulator. *arXiv preprint arXiv:2401.08022*, 2024.

[28] Joaquim Ortiz-Haro, Wolfgang Hoenig, Valentin N Hartmann, and Marc Toussaint. idb-a*: Iterative search and optimization for optimal kinodynamic motion planning. *arXiv preprint arXiv:2311.03553*, 2023.

[29] Mark Petersen and Russ Tedrake. Growing convex collision-free regions in configuration space using nonlinear programming. *arXiv preprint arXiv:2303.14737*, 2023.

[30] Ira Pohl. Heuristic search viewed as path finding in a graph. *Artificial intelligence*, 1(3-4):193–204, 1970.

[31] Nathan Ratliff, Matt Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *2009 IEEE international conference on robotics and automation*, pages 489–494. IEEE, 2009.

[32] Basak Sakcak, Luca Bascetta, Gianni Ferretti, and Maria Prandini. Sampling-based optimal kinodynamic planning with motion primitives. *Autonomous Robots*, 43(7):1715–1732, 2019.

[33] 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.

[34] Yuval Tassa, Nicolas Mansard, and Emo Todorov. Control-limited differential dynamic programming. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1168–1175. IEEE, 2014.

[35] Russ Tedrake and the Drake Development Team. Drake: Model-based design and verification for robotics, 2019. URL https://drake.mit.edu.

[36] Marc Toussaint. A tutorial on newton methods for constrained trajectory optimization and relations to slam, gaussian process smoothing, optimal control, and probabilistic inference. *Geometric and numerical foundations of movements*, pages 361–392, 2017.

[37] 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. *arXiv preprint arXiv:2310.02875*, 2023.

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

IX. APPENDIX

In the following sections, some of the definitions are repeated from the main portion of the paper for self-containedness.

Appendix 1

	Optimality	Bounded Suboptimality	Completeness
IxG	✗	✗	✓[5]
IxG*	✓	✓	✓

Optimality and completeness of IxG and IxG*.

Definition 1: **Markov Property in Search:** It states that the cost and the set of successors of a state depend *only* on the current state and not on the history of the path leading up to it.

As mentioned before, in the case of planning over a graph of convex sets, the graph is explicitly provided. So generating the set of successors of a state does not violate the Markov property. However, computing the cost of the successor requires computing the edge/trajectory and satisfying constraints which itself could depend on the path of convex sets leading up to the successor. This breaks the Markov property in IxG. We alleviate this in IxG* by allowing re-expansions with duplicates

Assumption 1: There exists a path in $G_{\mathcal{Q}}$ such that it contains $q^*_{0T}(t)$ minimizing Eq. 2.

A. Properties of IxG*

Theorem 1: **Optimality of IxG*:** If Assumption 1 holds, then on expanding $\mathcal{Q}^{0\dots T}$, *i.e.* when $\textsc{Key}(\mathcal{Q}^{0\dots T}) > \text{OPEN.min()}$ (Alg. 2, line 11), $\mathcal{Q}^{0\dots T}$.trajectory() will return $q^*_{0T}(t)$, where $q^*_{0T}(t)$ is the global minimizer of Eq. 2. Since this is proof of optimality, $\epsilon = 1$ in Alg. 2.

Proof: The proof is by induction. Let us assume that $\forall \mathcal{Q}^{0\dots b}$ which are expanded, $g(\mathcal{Q}^{0\dots b}) = g^*(\mathcal{Q}^{0\dots b})$. This is trivially true for $\mathcal{Q}_0$. We know that OPEN separates parts of the path from $\mathcal{Q}_0$ to goal $\mathcal{Q}_T$ that are expanded from the parts of the path that are never seen. The next state to be expanded is given by (Alg. 2, line 12 and 1)

$$\mathcal{Q}^{0\dots c} = \underset{\mathcal{Q}^{0\dots d} \in \text{OPEN}}{\arg\min}\ g(\mathcal{Q}^{0\dots d}) + l(\mathcal{Q}_d).$$

Let us assume $g(\mathcal{Q}^{0\dots d})$ is suboptimal. Then there must be at least one path $\mathcal{Q}^{0\dots e}$ in OPEN that is part of the optimal path from 0 to d on GCS which contains the optimal trajectory $q^*_{0d}(t)$. So

$$g(\mathcal{Q}^{0\dots e}) + l(\mathcal{Q}_e) \geq g(\mathcal{Q}^{0\dots c}) + l(\mathcal{Q}_c)$$

But

$$g(\mathcal{Q}^{0\dots e}) + c((\mathcal{Q}^{0\dots c}, \mathcal{Q}^{c\dots e})) < g(\mathcal{Q}^{0\dots c})$$

$$\Rightarrow g(\mathcal{Q}^{0\dots e}) + c((\mathcal{Q}^{0\dots c}, \mathcal{Q}^{c\dots e})) + l(\mathcal{Q}_e) < g(\mathcal{Q}^{0\dots c}) + l(\mathcal{Q}_c)$$

$$\Rightarrow g(\mathcal{Q}^{0\dots e}) + l(\mathcal{Q}_e) < g(\mathcal{Q}^{0\dots c}) + l(\mathcal{Q}_c)$$

[5]Under the assumption that all the edge constraints can be satisfied

The above step is a contradiction to our assumption that $g(\mathcal{Q}^{0\dots d})$ is suboptimal. Hence it must be the case that $g(\mathcal{Q}^{0\dots d})$ is optimal upon expansion. ■

Theorem 2: **Bounded suboptimality of IxG*:** If Assumption 1 holds, using $\epsilon > 1$ to prune duplicates under the criteria $c(q_{0s}(t)) + \epsilon * l(\mathcal{Q}_s) > \epsilon * u$ (Alg. 2, line 23), the termination condition $\textsc{Key}(\mathcal{Q}^{0\dots T}) > \text{OPEN.min()}$ (Alg. 2, line 11), will return trajectory $q^{\sim}_{0T}(t)$, whose cost $c(q^{\sim}_{0T}(t)) \leq \epsilon * c(q^*_{0T}(t))$ where $q^*_{0T}(t)$ is the global minimizer of Eq. 2.

Proof: The proof follows from the above proof of optimality applied to paths instead of vertices in the bounded suboptimality proof of wA* [30]. ■

Appendix 2

B. Properties of Lower Bound Graph (LBG)

The set of convex sets $\mathcal{Q} = \{\mathcal{Q}_1, \mathcal{Q}_2, \dots, \mathcal{Q}_n\} \subset \mathbb{R}^d$ are represented as a graph $G_{\mathcal{Q}} = (V_{\mathcal{Q}}, E_{\mathcal{Q}})$. Some of the properties of the lower bound graph $G_{lb} = (V_{lb}, E_{lb})$ include

- The number of vertices is bounded by
$$|\ V_{lb}\ | \leq 2 \sum_{i=1}^{n} |E^{in}_{\mathcal{Q}_i}||E^{out}_{\mathcal{Q}_i}|$$
where $E^{in}_{\mathcal{Q}_i}$ and $E^{out}_{\mathcal{Q}_i}$ are the incoming edges to and outgoing edges from vertex $\mathcal{Q}_i$.
- The number of edges is bounded by
$$|\ E_{lb}\ | \leq \frac{|\ V_{lb}\ |}{2} + \sum_{e_{\mathcal{Q}} \in E_{\mathcal{Q}}} |\ V^{e_{\mathcal{Q}}}_{lb}\ |^2$$
where $V^{e_{\mathcal{Q}}}_{lb} = \{v_{lb} \in V_{lb} \mid v_{lb} \in e_{\mathcal{Q}}\}$. An example of $v_{lb} \in e_{\mathcal{Q}}$ when the convex sets are overlapping can be $v_{lb} \in \mathcal{Q}_u \cap \mathcal{Q}_v$ where $e_{\mathcal{Q}} = (\mathcal{Q}_u, \mathcal{Q}_v)$.
- The degree of LBG is
$$deg(G_{lb}) = \max_{e_{\mathcal{Q}} \in E_{\mathcal{Q}}} |\ V^{e_{\mathcal{Q}}}_{lb}\ |$$

Let $\mathcal{M} : V_{lb} \rightarrow V_{\mathcal{Q}}$ denote a many-to-one mapping from the vertices in G_{lb} to the vertices in $G_{\mathcal{Q}}$. The mapping $\mathcal{M}^{-1}$ can be easily aggregated by keeping track of $v_{\mathcal{Q}} \in V_{\mathcal{Q}}$ for every $\mathcal{Q}_i \in V_{\mathcal{Q}}$ when constructing LBG (Alg. 3).

Theorem 3: Consider a sequence of convex sets $\mathcal{Q}_{1\dots K} = (\mathcal{Q}_1, \mathcal{Q}_2, \dots, \mathcal{Q}_K) \in V_{\mathcal{Q}}$ connected by edges from $E_{\mathcal{Q}}$. Let $q^*_{1K}(t)$ be the optimal trajectory via the sequence satisfying Eq. 2 and $G^{1\dots K}_{lb} = (V^{1\dots K}_{lb}, E^{1\dots K}_{lb})$ be a subgraph of G_{lb} corresponding to this sequence $\mathcal{Q}_{1\dots K}$. Let $v^k_{lb} \in \mathcal{M}^{-1}(\mathcal{Q}_k)$. Then the cost of the optimal path p^*_{1K} from v^1_{lb} to v^K_{lb} in the subgraph $G^{1\dots K}_{lb}$ underestimates $q^*_{1K}(t)$

$$c(p^*_{1K}) \leq c(q^*_{1K}(t))$$

Proof: Given a triplet sequence of convex sets, Alg. 3 constructs the LBG edges by solving the same minimization as the IxG* (Eq. 2) with a subset of constraints. For example, LBG construction does not have to satisfy some of the edge constraints of GCS such as continuity or smoothness. Therefore, by satisfying fewer constraints for the same sequence of convex sets, the cost of LBG solution is upper-bounded by the optimal solution in GCS. ■

Theorem 4: Consider $v_{lb}^s, v_{lb}^t \in V_{lb}$. Let $\Omega_s = \mathcal{M}(v_{lb}^s)$ and $\Omega_t = \mathcal{M}(v_{lb}^t)$. Let the optimal trajectory from Ω_s to Ω_t that satisfies Eq. 2 be $q_{st}^*(t)$. Then

$$c(r_{st}^*) \leq c(q_{st}^*(t))$$

where r_{st}^* is the optimal path on G_{lb} from v_{lb}^s to v_{lb}^t.

Proof: From the previous theorem, we know that the optimal path p_{st}^* in the subgraph $G_{lb}^{s\ldots t}$ holds

$$c(p_{st}^*) \leq c(q_{st}^*(t))$$

Since $G_{lb}^{s\ldots t}$ is a subgraph of G_{lb}

$$c(r_{st}^*) \leq c(p_{st}^*)$$

Thus

$$c(r_{st}^*) \leq c(q_{st}^*(t))$$

■