Multi Graph Search for High-Dimensional Robot Motion Planning

Itamar Mishani and Maxim Likhachev
Robotics Institute, School of Computer Science, Carnegie Mellon University
{imishani, maxim}@cs.cmu.edu

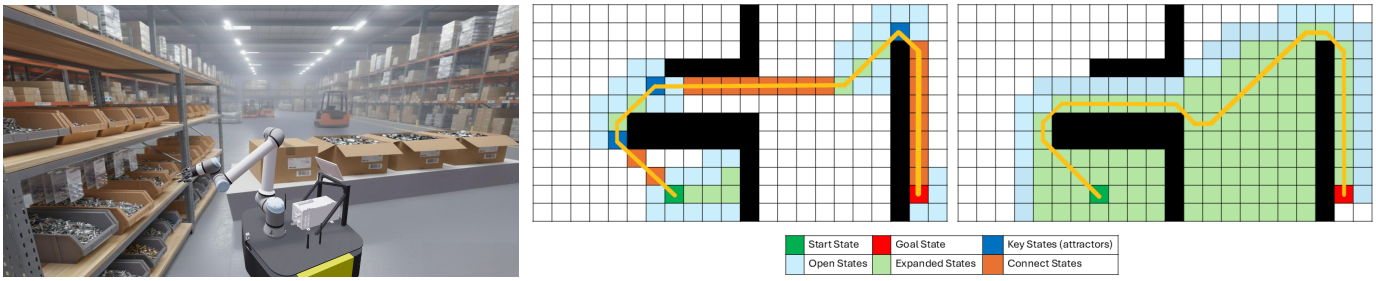


Fig. 1: Left: Mobile manipulators in warehouse settings demand efficient, predictable motion planning. Middle: MGS anchors search at key states and grows multiple subgraphs simultaneously, yielding a solution with a substantial reduction in search efforts. Right: Weighted-A* expands significantly more states to solve the same problem. Both searches operate on an 8-connected 2D grid with bounded suboptimality of 10.

***Abstract*—Efficient motion planning for high-dimensional robotic systems, such as manipulators and mobile manipulators, is critical for real-time operation and reliable deployment. Although advances in planning algorithms have enhanced scalability to high-dimensional state spaces, these improvements often come at the cost of generating unpredictable, inconsistent motions or requiring excessive computational resources and memory. In this work, we introduce *Multi-Graph Search* (MGS), a search-based motion planning algorithm that generalizes classical unidirectional and bidirectional search to a multi-graph setting. MGS maintains and incrementally expands multiple implicit graphs over the state space, focusing exploration on high-potential regions while allowing initially disconnected subgraphs to be merged through feasible transitions as the search progresses. We prove that MGS is complete and bounded-suboptimal, and empirically demonstrate its effectiveness on a range of manipulation and mobile manipulation tasks. Demonstrations, benchmarks and code are available at `<link omitted for review>`.**

I. Introduction

Collision-free motion planning is a fundamental problem in robotics, and a wide range of algorithms have been developed to address it [1, 2]. Despite this progress, reliably solving motion planning problems for high-dimensional robotic systems—such as mobile manipulators—remains challenging, particularly in high-stakes applications where fast, predictable, and consistent motion planners are essential for deployment (Fig. 1). A central difficulty lies in the inherent trade-off between planning efficiency and reliability. Sampling-based methods [3–5] achieve scalability by exploring the configuration space through randomly sampled collision-free configurations. While these approaches mitigate the computational burden in high-dimensional spaces by avoiding exhaustive exploration, their reliance on randomness often results in unpredictable behavior, inconsistent solutions, and limited guarantees beyond probabilistic guarantees. Such variability is undesirable in settings that demand repeatability and performance assurances. In contrast, search-based planning methods systematically expand states from the initial configuration toward the goal using heuristic guidance. These methods offer deterministic behavior, completeness, high consistency, and bounded suboptimality. However, their practical utility in high-dimensional systems is often hampered by the curse of dimensionality and a high sensitivity to heuristic accuracy.

This work is motivated by the observation that humans often solve complex tasks by identifying ”mental landmarks” [6, 7]—promising regions of the state space that guide efficient problem-solving. We leverage this intuition by introducing a planning algorithm that searches using a set of key states distributed throughout the state space. Unlike traditional search-based methods that progress unidirectionally from start to goal, our approach simultaneously explores multiple promising directions anchored by these key states, which serve as intermediate landmarks to structure and focus the search. This multi-directional search strategy enables efficient exploration of high-dimensional spaces while preserving the behavior characteristic and theoretical guarantees of classical search-based planners. In particular, our approach maintains completeness and bounded suboptimality, while substantially improving practical performance in high-dimensional environments.

Our main contributions are:

- A multi-directional heuristic search algorithm that per-

forms simultaneous searches rooted at a set of key states distributed throughout the configuration space.
- Theoretical analysis that establishes completeness and bounded suboptimality with respect to discretization guarantees for the proposed algorithm.
- A principled strategy for selecting the key states (i.e., graph roots) to identify promising regions for avoiding collisions.
- Extensive simulation experiments and real-world demonstrations showing improved performance and reliability over traditional baselines.

II. Related Work

We review methods for high-dimensional motion planning, focusing on scalability, consistency, and the use of intermediate states to guide search.

A. Classical Planning Approaches and Their Limitations

Sampling-based methods have become the standard for high-dimensional planning over the past two decades precisely because they scale well. Pioneering algorithms like Probabilistic Roadmaps (PRM) [4] and Rapidly-exploring Random Trees (RRT) [3] explore complex configuration spaces by random sampling, mitigating the curse of dimensionality inherent in exhaustive exploration. However, due to the stochastic nature of sampling-based algorithms, solutions are often inconsistent (similar scenarios are likely to have very different solutions) or appear unintuitive (i.e., highly suboptimal paths). In high-stakes industrial applications where reliability and repeatability are critical, this unpredictability presents challenges, leading practitioners to often rely on pre-recorded motions or restrictive assumptions [8–11].

In contrast, ***search-based methods*** offer strong theoretical properties—completeness, (bounded sub-) optimality, and consistency—making them appealing for reliable deployment. Graph search algorithms like A* [12] systematically expand states according to cost-to-come and heuristic estimates, with rigorous guarantees on solution quality. Variants of these methods have been successfully applied to robotic navigation [13–16] and extended to higher-dimensional systems including manipulation [2, 17–21], mobile manipulation [21, 22], and multi-arm systems [23, 24]. However, search-based methods struggle to scale to high dimensions due to the curse of dimensionality, requiring substantial engineering effort in heuristic design and pruning to control computational complexity.

Optimization-based methods offer a different perspective, formulating planning as trajectory optimization subject to constraints. Approaches like CHOMP [25], STOMP [26], GPMP2 [27], and TrajOpt [28] leverage numerical optimization to generate smooth, dynamically feasible motions. Yet these methods lack completeness or convergence guarantees and are highly sensitive to initialization, frequently converging to local minima in cluttered environments.

Recently, ***learning-based methods*** have emerged as a promising direction, using neural networks to learn planning strategies from data [29–32]. Such approaches aim to leverage learned representations to plan, accelerate planning, or improve sampling efficiency. While these methods show promise in reducing planning time through learned priors, they generally lack the completeness and bounded suboptimality guarantees essential for safety-critical applications, and their generalization to novel environments remains a challenge.

B. Bidirectional and Multi-Directional Search Strategies

A key insight from both sampling-based and search-based planning is that exploring from multiple directions can dramatically improve efficiency. In sampling-based planning, bidirectional variants like RRT-Connect [33] accelerate planning by simultaneously growing trees from start and goal, greedily attempting connections between frontiers to efficiently navigate narrow passages and reduce exploration redundancy. The success of RRT-Connect demonstrates that connecting two exploration frontiers can be far more efficient than unidirectional search. Extensions to multi-directional sampling-based methods [34] have been explored, but rely on random selection of additional root states, inheriting the unpredictability characteristic of sampling-based approaches. For search-based planning, bidirectional search strategies have been explored where frontiers expand simultaneously from start and goal [35, 36]. While bidirectional heuristic searches often struggle with frontier alignment (the "missile metaphor" [35]), approaches like A*-Connect [37] actively guide convergence while preserving suboptimality bounds [38].

The success of bidirectional search naturally motivates exploring from multiple frontiers, yet generalizing beyond two frontiers faces key technical challenges: identifying where to root searches is non-trivial (randomly sampling key states [34] reintroduces the unpredictability search-based methods aim to avoid), coordinating multiple frontiers to ensure completeness and bounded suboptimality is algorithmically challenging, and efficiently merging disconnected subgraphs requires careful algorithmic design. These challenges help explain why multi-directional search has remained largely unexplored in search-based motion planning.

C. Subgoal-Guided and Region-Based Planning

The intuition of guiding planners through high-potential regions or intermediate subgoals has been explored across multiple research communities. In the heuristic search literature, landmark-based planning [39] has been widely employed to enhance heuristic estimates and decompose complex problems into manageable sequences of subgoals [40–43]. However, these classical approaches typically enforce that landmarks be satisfied in a fixed order as part of the final solution, making them distinct from MGS, which treats key states as *optional* seeds for simultaneous multi-directional exploration rather than mandatory waypoints.

Similarly, in reinforcement learning, subgoals have been used to structure hierarchical learning and enable faster exploration [44–46]. Works in hierarchical RL leverage subgoals to decompose tasks into manageable subtasks, allowing agents to learn reusable skills and accelerate convergence. This

principle—that strategically chosen intermediate objectives can dramatically improve search efficiency—is equally applicable to motion planning. By identifying and leveraging key states distributed throughout the configuration space, we can focus exploration toward promising regions and substantially accelerate planning, analogous to how subgoals guide learning in RL.

Translating this intuition into practice requires principled methods for identifying key states or promising regions. Recently, methods in search-based planning [47, 48] and optimization-based planning [49] propose to decompose the environment into promising regions (e.g., convex regions or regions centered around key states) to facilitate more efficient planning and have shown to be especially effective in industrial settings. Yet these methods require substantial preprocessing and strong assumptions about environment structure, limiting applicability in dynamic settings.

MGS enables multi-directional search-based planning with principled online key state selection, combining scalability with theoretical guarantees.

III. Preliminaries

We begin by formally defining the motion planning problem and its graph-based representation. Next, we introduce focal search, a bounded suboptimal search technique that enables efficient exploration through inadmissible heuristics, forming the basis for our suboptimality guarantees. Finally, we review attractor-based methods and introduce notations for root selection, providing the foundation for identifying strategic key states through workspace reasoning.

A. Problem Formulation

We consider collision-free motion planning for a robot $\mathcal{R}$ with configuration space $\mathcal{C} \subseteq \mathbb{R}^d$, where d denotes the number of degrees of freedom. The configuration space is partitioned into free space $\mathcal{C}_{\text{free}}$ and obstacle space $\mathcal{C}_{\text{obs}}$, such that $\mathcal{C} = \mathcal{C}_{\text{free}} \cup \mathcal{C}_{\text{obs}}$ and $\mathcal{C}_{\text{free}} \cap \mathcal{C}_{\text{obs}} = \emptyset$. The robot operates in a world $\mathcal{W} \subseteq \mathbb{R}^3$, with end effector workspace $\mathcal{W}_{\text{eff}} \subseteq \mathcal{W}$ and task space $\mathcal{T} \subseteq SE(3)$. A motion planning problem instance is defined by a start configuration $q_{\text{start}} \in \mathcal{C}_{\text{free}}$ and a goal condition (predicate) $\phi_{\text{goal}} : \mathcal{C} \to \{0, 1\}$ that specifies the set of satisfying goal configurations $\mathcal{C}_{\text{goal}} = \{q \in \mathcal{C}_{\text{free}} \mid \phi_{\text{goal}}(q) = 1\}$. In this work, we consider goal conditions defined either in configuration space or in the end effector task space (e.g., $\phi_{\text{goal}}(q) = 1$ if $FK(q) \in \mathcal{T}_{\text{goal}}$, where $\mathcal{T}_{\text{goal}} \subseteq SE(3)$ is a desired task space region). The objective is to find a collision-free path $\pi : [0, 1] \to \mathcal{C}_{\text{free}}$ such that $\pi(0) = q_{\text{start}}$ and $\phi_{\text{goal}}(\pi(1)) = 1$, while minimizing a cost function $c(\pi)$.

Graph Search Representation. To leverage graph search techniques, we discretize the configuration space $\mathcal{C}$ into a graph $G = (V, E)$, where V is the set of vertices representing discrete configurations and E is the set of edges representing feasible transitions between configurations. Each vertex $v \in V$ corresponds to a configuration $q_v \in \mathcal{C}_{\text{free}}$. Edges are defined by applying motion primitives $\mathcal{M} = \{m_1, \ldots, m_k\}$—predefined local motions—and an edge $(u, v) \in E$ is added only if the motion primitive connecting q_u to q_v is collision-free, as verified through collision checking. The cost of traversing an edge (u, v) is denoted by $c(u, v)$, typically defined as uniform cost or as the length of the motion primitive. Since it is not feasible to enumerate all configurations in high-dimensional spaces, we construct an implicit graph where vertices and edges are generated on-the-fly during the search process as it expands already encountered states and applies motion primitives to generate new states.

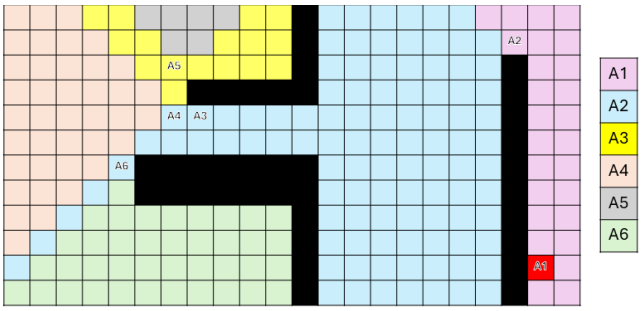


Fig. 2: Illustration of attractor states (A_i) and their regions of trivial connectivity. States within each region can reach the corresponding attractor through greedy tracing. The red cell is the state from which we expand the BFS wavefront. The grid is 8-connected with cardinal cost 1, diagonal cost $\sqrt{2}$, and Euclidean distance as the potential.

B. Focal Search for Sub-graph Exploration

Focal search [50] is a bounded suboptimal search technique that enables efficient exploration by relaxing strict optimality requirements. In standard A* search, states are expanded in order of their f-value, where $f(s) = g(s) + h(s)$, with $g(s)$ being the cost-to-come and $h(s)$ an admissible heuristic estimate of the cost-to-go. While this guarantees optimality, it can be inefficient when the heuristic provides weak guidance in complex environments.

Focal search introduces a bounded relaxation by maintaining two priority queues: `OPEN` (ordered by f-value) and `FOCAL` (a subset of `OPEN`). Given a suboptimality bound $\epsilon \geq 1$, `FOCAL` contains all states $s \in$ `OPEN` with $f(s) \leq \epsilon \cdot f_{\min}$, where $f_{\min}$ is the minimum f-value in `OPEN`. During expansion, focal search selects states from `FOCAL` according to an alternative criterion, often an inadmissible heuristic $\hat{h}(s)$ that provides stronger guidance than the admissible heuristic $h(s)$. In our multi-graph search framework, focal search is used to provide bounded suboptimality guarantees while exploring individual sub-graphs rooted at key states (roots).

C. Attractor-Based Methods for Root Selection

The concept of *attractor states* was originally introduced for constant-time motion planning [47] and memory-efficient search [51]. In these works, attractors serve as representative states around which neighborhoods are organized, enabling efficient path reconstruction without storing complete paths explicitly. The key mechanism underlying attractors is *greedy*

tracing, which allows states to reach attractors by iteratively following predecessors that minimize a potential function.

Given a search space $\mathcal{S}$ and a non-negative function $h_{\text{potential}} : \mathcal{S} \times \mathcal{S} \to \mathbb{R}_{\geq 0}$ (e.g., Euclidean distance), greedy tracing is formally defined as follows:

Definition 1 (Greedy Predecessor)**.** *For a state $s \in \mathcal{S}$ and target state $s_{target} \in \mathcal{S}$, a predecessor $s' \in \text{PRED}(s)$ is the* greedy predecessor *of s with respect to s_{target} if*

$$s' = \arg \min_{s'' \in \text{PRED}(s)} h_{potential}(s'', s_{target})$$

Note that a tie-breaking rule is applied to ensure uniqueness of the greedy predecessor.

Definition 2 (Greedy Tracing)**.** *Greedy tracing with respect to $h_{potential}$ from state s toward target state s_{target} iteratively selects the greedy predecessor of s with respect to s_{target}, continuing until reaching s_{target}.*

An attractor state $a \in \mathcal{S}$ is a state whose neighborhood can reach it via greedy tracing, defining a region of *trivial connectivity* (Fig. 2).

IV. Multi-Graph Search

In this section, we present MGS (**M**ulti-**G**raph **S**earch) by formalizing the multi-directional problem, introducing the anchor and connect heuristics for exploration and connectivity, describing the algorithm mechanics, and presenting our workspace-aware root selection method.

A. Multi-Directional Problem Formulation

Traditional search-based planners expand states from a single start configuration q_{start} toward the goal condition ϕ_{goal}. This unidirectional approach can be inefficient in high-dimensional spaces, as it may explore large portions of the state space that do not contribute to finding a solution. To address this limitation, MGS employs a multi-directional search strategy that concurrently explores multiple search sub-graphs rooted at a set of key states (roots) $\{r_1, r_2, \ldots, r_m\}$ distributed throughout the configuration space. Formally, instead of maintaining a single implicit graph $G = (V, E)$, we construct and explore a collection of sub-graphs, $\mathcal{G} = \{G_1, G_2, \ldots, G_m\}$, where each sub-graph $G_i = (V_i, E_i)$ is an undirected graph rooted at key state $r_i \in \mathcal{C}_{\text{free}}$. The roots include the start configuration ($r_1 = q_{\text{start}}$), intermediate key states (Section IV-D), and optionally the goal configuration ($r_m = q_{\text{goal}}$) when specified[1]. We distinguish between two types of searches: the *anchor search* G_1 rooted at $r_1 = q_{\text{start}}$ uses focal search (Section III-B) with suboptimality bound $\epsilon \geq 1$ to explore toward the goal, while *connect searches* $G_2, \ldots, G_m$ use single-queue search (`OPEN` only) to facilitate connections between regions of the configuration space.

Each sub-graph grows independently—the anchor search G_1 expands states from its `FOCAL` queue, while connect searches $G_2, \ldots, G_m$ expand from their respective `OPEN` queues. A solution is found when there exists a connected path through the union of sub-graphs from q_{start} to a configuration satisfying ϕ_{goal}. Formally, we seek a path π that traverses connected sub-graphs:

$$\pi = \pi_1 \oplus \pi_2 \oplus \cdots \oplus \pi_k$$

where each π_i is a path segment within a sub-graph $G_{j_i} \in \mathcal{G}$, π_1 starts at q_{start} in G_1, consecutive segments connect at shared vertices, and $\phi_{\text{goal}}(\pi(1)) = 1$.

[1]For multi-goal problems, each goal configuration can serve as a root, up to a maximum of m sub-graphs. Here, we assume a single goal configuration or end-effector pose.

B. Search Heuristics

The efficiency of MGS relies on complementary heuristics that guide the anchor and connect searches:

Anchor Search Heuristics. The anchor search G_1 uses focal search with two heuristics: `OPEN` is ordered by $f(s) = g(s) + h(s)$ where h is admissible (e.g., joint Euclidean distance), and `FOCAL` is ordered by an inadmissible heuristic $\hat{h}(s)$ (task space distance) as described in Section III-B.

Connect Search Heuristics. Each connect search G_i ($i \geq 2$) uses a single heuristic based on distance to the other sub-graphs:

$$h_{\text{connect}}(s) = \min_{s' \in \text{FRONTIER}(\mathcal{G} \setminus \{G_i\})} d(s, s')$$

Where the distance function $d(s, s')$ is a pairwise heuristic (e.g., Euclidean distance in configuration space or workspace) measuring proximity between states s and s', and $\text{FRONTIER}(\mathcal{G} \setminus \{G_i\})$ denotes the set of states across all the `OPEN` lists of the sub-graphs except G_i. In other words, $h_{\text{connect}}(s)$ corresponds to the Front to Front (F2F) heuristic [52], guiding connect searches to prioritize expanding states closest to the frontiers of other sub-graphs.

C. Multi-Graph Search Algorithm

Algorithm 1 presents MGS, our multi-directional graph search algorithm. MGS initializes by selecting root configurations (Lines 1–2, see Section IV-D) and constructing an anchor search G_1 rooted at q_{start} with `OPEN` and `FOCAL` queues, and connect searches $G_2, \ldots, G_m$ each with a single `OPEN` queue.

The main loop (Line 8) alternates between two phases. In Phase 1 (Anchor Expansion), the anchor search expands the best state from its `FOCAL` queue according to $\hat{h}$ (Line 9). If this state satisfies the goal condition, a solution path is returned (Lines 10–11). Otherwise, the algorithm attempts to connect to nearby frontier states of other sub-graphs via collision-free edges (Line 12). Successful connections trigger merging into G_1 (Line 16). The merging procedure (`MergeSubGraphs`) re-roots the merged sub-graph at the connection state, integrates all its vertices and edges into the anchor graph, and propagates updated g-values and updates h-values accordingly, adding newly merged states to the anchor's `OPEN` and `FOCAL` queues. States that were closed in the merged sub-graph are added to the anchor's `OPEN` queue, allowing re-expansion in the anchor search. Additionally, if the expanded state was already closed in another sub-graph, the two are merged

Algorithm 1: Multi-Graph Search (MGS)

INPUT: Start configuration $q_{\text{start}} \in \mathcal{C}_{free}$
Goal termination condition function $\Phi_{goal} : \mathcal{C} \to \{0, 1\}$
Maximum number of sub-graphs m
Suboptimality bound $\epsilon \geq 1$
Admissible heuristic $h : \mathcal{C} \to \mathbb{R}_{\geq 0}$
Inadmissible focal heuristic $\hat{h} : \mathcal{C} \to \mathbb{R}_{\geq 0}$
Connect Heuristic $h_{\text{connect}} : \mathcal{C} \times \mathcal{C} \to \mathbb{R}_{\geq 0}$
OUTPUT: Collision-free trajectory from q_{start} to a goal configuration satisfying Φ_{goal}

```
// Initialize sub-graphs rooted at key states
1  roots = GetRoots (q_start, Φ_goal, m − 1)
2  G ← InitializeSubGraphs(roots)
3  G_1.InitializeFocalSearch (ε, h, ĥ)
4  G_1.AddVertex (q_start, g = 0) ;  // Initialize start with zero cost
5  for i ∈ 2 to |G| do
6      G_i.InitializeSearch (h_connect)
7      G_i.AddVertex (r_i, g = 0) ; // Initialize root with zero cost
   // Main search loop: each iteration expands one anchor state and all connect states
8  while ¬G_1.FOCAL().IsEmpty() ∧ ¬ TimeOut() do
       // Phase 1: Anchor expansion
9      q_a ← G_1.FOCAL().pop()
10     if Φ_goal(q_a) then
11         return G_1.ReconstructPath (q_start, q_a)
       // Try to connect to other graphs
12     if TryToConnect (q_a, G \ {G_1}) then
13         for G_connected ∈ GraphsConnectedTo (q_a, G) do
14             π ← GetConnectingPath (G_1, G_connected, q_a)
15             G_1.AddVerticesAndEdges (π)
16             G ← MergeSubGraphs (G_1, G_connected, π[|π|]) ;
                   // Anchor always receives merge
       // Check if already expanded in another graph, else expand
17     if q_a ∈ ⋃_{j=2}^{|G|} G_j.CLOSED() then
18         G_closed ← FindGraphClosingState (q_a, G)
19         G ← MergeSubGraphs (G_1, G_closed, q_a)
20     else
21         G_1.ExpandState (q_a)
       // Phase 2: Connect expansions
22     for i ∈ 2 to |G| do
23         if ¬G_i.OPEN().IsEmpty() then
24             q_c ← G_i.OPEN().pop()
25             if TryToConnect (q_c, G \ {G_i}) then
26                 for G_connected ∈ GraphsConnectedTo (q_c, G) do
27                     G_from, G_to ← ChooseMergingOrder (G_i, G_connected, G)
28                     π ← GetConnectingPath (G_from, G_to, q_c)
29                     G_from.AddVerticesAndEdges (π)
30                     G ← MergeSubGraphs (G_from, G_to, π[|π|])
31             if q_c ∈ ⋃_{j=1,j≠i}^{|G|} G_j.CLOSED() then
32                 G_closed ← FindGraphClosingState (q_c, G)
33                 G ← MergeSubGraphs (G_i, G_closed, q_c)
34             else
35                 G_i.ExpandState (q_c)
36 return Failure
```

immediately without requiring an explicit edge connection (Line 17).

In Phase 2 (Connect Expansions, Line 22), each connect search G_i ($i \geq 2$) expands its best state according to h_{connect}, which prioritizes states near frontier states of other sub-graphs. Connection attempts and potential merges follow similar logic to Phase 1 (Lines 25–30), with merging order determined by `ChooseMergingOrder` (Line 27) based on which sub-graph is closer to the anchor search. When two connect searches merge (neither being the anchor), their `OPEN` queues are combined but closed states are not reopened—the merged sub-graph inherits the closed sets of both, avoiding redundant expansions until the eventual merge into the anchor. The algorithm terminates when either a solution is found via the anchor search (Line 11) or when the anchor's `FOCAL` queue becomes empty (Line 8), indicating no ϵ-suboptimal solution exists. Fig. 1 illustrates this on a 2D grid: MGS yields a solution with significantly fewer expansions than weighted A* under the same suboptimality bound.

Algorithm 2: Workspace-Aware Root Selection via Backward BFS

INPUT: Start configuration $q_{\text{start}} \in \mathcal{C}_{free}$
Goal termination condition function $\Phi_{goal} : \mathcal{C} \to \{0, 1\}$
Maximum number of sub-graphs m
OUTPUT: Set of root configurations $\{r_1, r_2, \ldots, r_m\}$

```
1  NOTE: Workspace W_eff is discretized into a 3D occupancy grid
2  GetRoots (q_start, Φ_goal, m)
3      if goal configuration is fully specified then
4          roots ← {Φ_goal.GetRobotConfiguration()} ;
               // Include goal as root
5      else
6          roots ← ∅
7      w.state ← Φ_goal.GetRobotEEgoalPosition() ;  // Start from goal end effector position
8      w.g ← 0 ;                                   // Zero cost at goal
9      w.attractor ← w.state ; // The BFS start state is the first attractor
10     CLOSED ← ∅
11     OPEN ← {w} ;  // Initialize with start state (FIFO)
12     ATTRACTORS ← ∅
13     while ¬ OPEN.IsEmpty() do
14         w ← OPEN.pop()
15         Insert w into CLOSED
16         foreach neighbor w′ of w do
17             if w′ is not in collision ∧ w′ ∉ CLOSED then
18                 if w′.g > w.g + cost(w, w′) then
19                     w′.g ← w.g + cost(w, w′) ;      // We use uniform cost
20                     a ← w.attractor ;      // Get attractor from parent
21                     greedy_pred ← arg min_{w″∈PRED(w′)} d(w″, a) ;
                           // Greedy predecessor toward attractor
22                     if greedy_pred ≠ w then
23                         w′.attractor ← w.state ;      // New attractor found at this branching point
24                         if w.state ∉ ATTRACTORS then
25                             Insert (w.state, GetRobotConfiguration(w)) into ATTRACTORS
26                     else
27                         w′.attractor ← a ;      // Inherit attractor from parent
28                 if w′ ∉ OPEN then
29                     Insert w′ into OPEN
       // Adding the forward attractors when following the computed policy from start to goal
30     forward_attractors ← ForwardAttractors (q_start, Φ_goal.GetRobotEEgoalPosition())
31     foreach (w_attr, q_attr) ∈ forward_attractors do
32         if (w_attr, q_attr) ∉ ATTRACTORS then
33             ATTRACTORS ← ATTRACTORS ∪ {(w_attr, q_attr)}
       // Cluster attractors to select up to maximum of m roots
34     if |ATTRACTORS| + |roots| ≤ m then
35         roots ← roots ∪ {q_attr | (w_attr, q_attr) ∈ ATTRACTORS}
36     else
37         clusters ← Cluster (ATTRACTORS, m − |roots|)
38         foreach cluster c ∈ clusters do
39             q_rep ← representative configuration of cluster c
40             roots ← roots ∪ {q_rep}
41     return roots
```

D. Workspace-Aware Root Selection via Backward BFS

Selecting roots $\{r_1, r_2, \ldots, r_m\}$ is critical to MGS's efficiency. We seek a principled method for selecting intermediate states that are likely to lie along useful paths from start to goal. Identifying such waypoints directly in the high-dimensional configuration space $\mathcal{C}$ is computationally expensive. Instead, we approximate the collision-free regions in the lower-dimensional end effector workspace $\mathcal{W}_{\text{eff}}$ by discretizing it into a 3D occupancy grid based on obstacle geometry. We then employ backward breadth-first search (BFS) in this discretized workspace to identify intermediate *attractor* states that mark the boundaries of distinct collision-free regions [51]. These workspace attractors are mapped back to configuration space to serve as roots. Alg. 2 outlines the procedure.

The algorithm proceeds in three stages: backward 3D BFS to compute a workspace policy and identify attractors, forward rollout to select attractors along the start-to-goal path, and clustering to respect the roots budget. The BFS is initialized from the goal end effector position in the discretized workspace grid, with zero cost and itself as the initial attractor (Line 9). If the goal configuration is fully specified, it is also included as a root (Line 6). The core insight is that *attractors* emerge at workspace locations where the shortest path from goal diverges due to obstacles (Fig. 2). As the BFS wavefront expands backward from the goal (Line 13), each state w maintains a reference to its current attractor—the workspace position it is "attracted toward" when following a greedy path to the goal. When expanding a neighbor w', the algorithm checks whether the greedy predecessor toward the current attractor a matches the actual BFS parent w (Line 21). If they differ, this indicates a branching point where the obstacle geometry forces paths to diverge, and w becomes a new attractor (Line 23). Otherwise, w' inherits the attractor from its parent. Each attractor is stored along with a corresponding robot configuration obtained via differential inverse kinematics, seeded with the configuration of the previous attractor. The backward BFS identifies attractors with respect to the goal, but we also want to identify important attractors with respect to the start. After the BFS completes, the algorithm traces the computed policy forward from the start position to the goal, collecting the attractors encountered along this path (Line 30). A further discussion and intuition on forward attractors appears in the appendix. The forward attractors are added to the attractor set, and if the total number of attractors does not exceed the budget m, all are included as roots (Line 34). Otherwise, spatial clustering (e.g., k-means in workspace) groups nearby attractors, and a representative configuration from each cluster is selected as a root (Line 37). This ensures the roots remain well-distributed while respecting the computational budget. The resulting root configurations $\{r_1, \ldots, r_m\}$ capture strategic waypoints that the robot's end effector must navigate around obstacles, providing MGS with informed starting points for its connect searches.

E. Theoretical Properties

We establish that MGS inherits the completeness and bounded suboptimality guarantees of focal search. The following properties hold with respect to the implicit graph $G = (V, E)$ induced by the discretization described in Section III.

Theorem 1 (Bounded Suboptimality)**.** *Let $h : \mathcal{C} \to \mathbb{R}_{\geq 0}$ be an admissible heuristic (i.e., $h(q) \leq c^*(q, q_{goal})$ for all $q \in \mathcal{C}$). If* MGS *returns a solution path π, then cost$(\pi) \leq \epsilon \cdot c^*$, where c^* is the optimal solution cost and $\epsilon \geq 1$ is the suboptimality bound.*

Proof Sketch: Solutions are returned exclusively through the anchor search G_1 when a state $q_a \in$ FOCAL satisfies $\Phi_{\text{goal}}(q_a) = 1$ (Alg. 1, Line 9). The anchor maintains the focal search invariant: all states in FOCAL have $f(s) \leq \epsilon \cdot f_{\min}$, where $f_{\min} = \min_{s \in \text{OPEN}} f(s)$. Since h is admissible, $f_{\min} \leq c^*$ throughout the search. When connect searches merge into the anchor via MergeSubGraphs, the g-values of merged states are recomputed with respect to q_{start} by propagating costs through the connecting path. This preserves the focal invariant: merged states are inserted into OPEN with correct g-values, and only enter FOCAL if their f-values satisfy the bound. Since the returned solution has $g(q_a) = \text{cost}(\pi)$ and q_a was in FOCAL, we have $\text{cost}(\pi) = f(q_a) \leq \epsilon \cdot f_{\min} \leq \epsilon \cdot c^*$. ∎

Theorem 2 (Bounded Re-Expansions)**.** *If h is consistent (i.e., $h(q) \leq c(q, q') + h(q')$ for all edges (q, q')), then each state is expanded at most twice during* MGS *execution.*

Proof Sketch: With a consistent heuristic, focal search never re-expands states within a single sub-graph. A state q may be expanded once in a connect search G_i before merging, and at most once more in the anchor G_1 after merging (since merged states are added to OPEN and may be re-expanded with updated g-values). Once expanded in the anchor with the correct g-value, consistency ensures q will not be expanded again. ∎

Theorem 3 (Completeness)**.** *If a solution exists,* MGS *will find one (given sufficient time and memory).*

Proof Sketch: The anchor search G_1 performs focal search, which is complete: it systematically expands states from FOCAL and will eventually expand all reachable states if no solution is found earlier. Connect searches and merging only accelerate the search by adding states to the anchor's OPEN queue—they never remove states or prevent exploration. If a solution path exists from q_{start} to a goal state, the anchor will eventually either (a) discover it directly through expansion, or (b) discover it through states added via merging. In either case, the goal state will eventually enter FOCAL and be returned. ∎

V. EXPERIMENTS

Our experimental design focuses on two core objectives: demonstrating that MGS can efficiently generate high-quality solutions for complex motion planning tasks, and validating

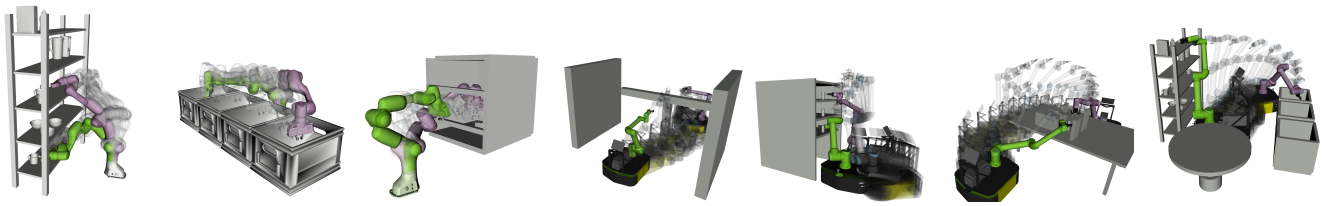


Fig. 3: Experimental environments for manipulation (left three: shelf pick-and-place, bin picking, cage extraction) and mobile manipulation (right four: low-clearance passage, deep shelf reach, cluttered table, combined warehouse).

the algorithm's consistency across repeated and perturbed trials. We compare MGS against state-of-the-art sampling-based baselines provided by OMPL [1] and optimization-based baselines, as well as search-based planners via the SRMP framework [2].

A. Experimental Setup

We used scenarios from the Motion Benchmarker [53] and extended it with additional environments representing realistic industrial challenges such as narrow passages, cluttered workspaces, and confined reaching tasks. The evaluation covers seven environments spanning manipulation and mobile manipulation (Fig. 3). We validate performance on two hardware configurations: a Franka Emika Panda (7-DOF) for fixed-base manipulation and a Ridgeback omni-directional base mounted with a UR10e arm (9-DOF total) for mobile manipulation.

For the general benchmark, we generated a dataset of 50–150 unique planning queries per scenario. Each specific query was solved 5 times by every planner to assess repeatability and performance variance. In addition to standard benchmarking, we incorporated a specific robustness test to evaluate solution consistency. In this test, each base problem instance was perturbed 10 times by applying small random noise to valid start and goal states. A consistent and predictable planner should yield similar performance across these perturbations.

Experiments were conducted on an Intel Core i9-12900H laptop (64GB RAM, 5.2GHz). All algorithms were implemented in C++ using MoveIt! [54] and the Flexible Collision Library (FCL) [55] for validity checking. A 5-second time limit was enforced for all runs. Aggregated results are presented below, differentiated by task type (manipulation and mobile manipulation). Detailed breakdowns per scenario and additional experiments and ablation studies are provided in the appendix and on the project website.

B. Evaluation Metrics

We assess performance using four primary metrics: *success rate*, *cost*, *consistency*, and *planning time*. Path cost is calculated as a weighted sum of length, velocity, and acceleration:

$$\text{cost} = L + 0.1 \cdot V + 0.01 \cdot A$$

where L, V, and A represent the cumulative joint-space distance, velocity magnitude, and acceleration magnitude, respectively. This composite metric ensures a balance between trajectory efficiency and execution smoothness.

To measure consistency, we calculate the *coefficient of variation* (CV) as a percentage. For a set of path costs $\{c_1, c_2, \ldots, c_n\}$ obtained from n repetitions, the CV is the ratio of the standard deviation to the mean ($\text{CV} = \sigma/\mu$). Lower CV values indicate high consistency, whereas higher values reflect significant variance in solution quality.

Finally, we report the average *planning time* across all trials and the *success rate*, representing the percentage of queries successfully solved within the allowed planning time.

C. Evaluated Algorithms

We compare MGS against 12 baselines spanning three paradigms. **Sampling-based:** RRT [3], RRT-Connect [33], BiTRRT [56], BiEST [57], PRM [4], and RRT* [5], with shortcutting (OMPL implementation) and time parameterization as postprocessing. We exclude MTRRT [58] from the comparison: we could not find an open-source implementation, and our own implementation failed to solve most of the benchmark problems in this experimental suite. **Optimization-based:** CHOMP [25], STOMP [26], and a hybrid RRT-Connect+CHOMP pipeline. CHOMP and STOMP were initialized with straight-line joint-space trajectories, while RRT-Connect+CHOMP used RRT-Connect to generate an initial feasible path that was subsequently refined by CHOMP. All optimization-based planners used the MoveIt! plugin implementations. **Search-based:** Weighted-A* (wA*), MHA* [21], and wPA*SE [59], using joint-space Euclidean and workspace 3D BFS heuristics, with shortcutting (as in [2]) and the same time parameterization as the sampling-based methods. For all search-based planners, we used uniform-cost edges with bounded suboptimality $w = 50$[2]. MGS has the same heuristic settings as the search-based baselines, same edge costs and suboptimality bound, and a maximum of 10 subgraphs.

D. Results and Analysis

Table I summarizes the aggregated results across all manipulation and mobile manipulation tasks. MGS consistently attains high success rates while producing low-cost solutions. Among sampling-based methods, BiTRRT achieves comparable reliability but at noticeably higher path costs. Search-based planners achieve similar solution quality to MGS but suffer from lower success rates, particularly in mobile manipulation

[2]The heuristic underestimates the cost-to-go and edges have unit cost; the weight w scales the heuristic to match edge transition costs and further inflates it.

TABLE I: Experimental Results Summary: Aggregated performance metrics for manipulation and mobile manipulation tasks. Pairwise Relative Cost is the ratio of MGS's path cost to each baseline, computed only on queries where both planners succeeded. Planning times are reported for successful runs and averaged over all runs (including timeouts).

			MGS	WA*	MHA*	wPASE	RRT	PRM	RRT-Connect	BiTRRT	BiEST	CHOMP	STOMP	RRT-Connect + CHOMP	RRT*
Manipulation	Success Rate [%]		98.2	87.5	87.5	90.4	65.7	81.4	86.8	97.5	83.6	12.5	77.1	65.7	68.2
	Pairwise Relative Cost [MGS / *planner*]		-	0.98	0.99	0.99	0.74	0.74	0.78	0.81	0.77	0.47	0.59	0.47	1.11
	Planning Time [sec]	Successful Runs	0.41	0.26	0.14	0.19	0.86	0.23	0.37	0.39	0.48	0.09	0.25	0.14	5
		All Runs (including time-limit)	0.49	1.18	1.15	0.94	2.61	1.62	1.27	0.56	1.64	4.43	1.75	2.19	5
Mobile Manipulation	Success Rate [%]		100	74.4	88.5	84.9	74.5	94.8	100	100	97.7	22.4	55.7	84.6	63.3
	Pairwise Relative Cost [MGS / *planner*]		-	1.07	1.10	1.10	0.82	0.77	0.80	0.88	0.72	0.66	0.98	0.72	0.90
	Planning Time [sec]	Successful Runs	0.20	0.20	0.32	0.37	0.39	0.35	0.11	0.21	0.24	0.18	0.49	0.24	5.00
		All Runs (including time-limit)	0.20	1.59	0.86	1.16	1.48	0.58	0.11	0.21	0.46	4.14	2.75	1.03	5.00

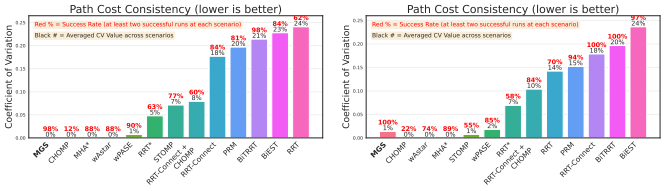


Fig. 4: Path cost consistency for manipulation (left) and mobile manipulation (right) tasks. Each planner was executed 5 times per query; bars show the coefficient of variation (CV) of path costs across runs.

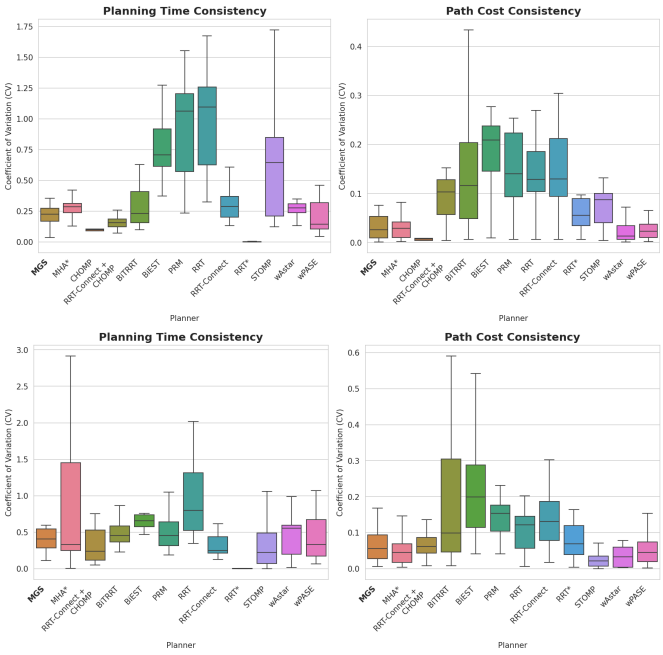


Fig. 5: Consistency analysis under start/goal perturbations for manipulation (top) and mobile manipulation (bottom) tasks. Each planner was tested on 10 perturbed versions of each base query; bars show the coefficient of variation (CV) of path costs and planning times across perturbations.

where heuristic guidance becomes less effective in higher-dimensional spaces. Optimization-based approaches (CHOMP, STOMP) show the lowest reliability due to frequent convergence to infeasible local minima in cluttered environments. The only method with lower costs than MGS on successful queries is RRT*, which is expected given its asymptotic optimality guarantee; however, this comes at the expense of significantly lower success rates and longer planning times.

Figures 4 and 5 present the consistency analysis across repeated runs and under start/goal perturbations. MGS exhibits low coefficients of variation (CV) in path costs, indicating stable and predictable performance. This consistency stems from the deterministic nature of the search-based anchor combined with structured root selection. In mobile manipulation, slightly higher variability is observed, as the differential IK used for root selection (Section IV-D) can produce different configurations depending on the seed, leading to occasional variation in the resulting roots. In contrast, sampling-based planners show higher variability due to stochastic exploration, where small perturbations in start/goal can lead to substantially different random trees.

MGS's performance depends on the quality and on the number of roots: too few roots can limit the benefit of multi-directional, while too many dilute the expansion budget across subgraphs that may not contribute to the solution. Additional analysis of these sensitivities is provided in the appendix. Overall, MGS effectively bridges the gap between sampling-based and search-based paradigms—achieving the scalability of the former while maintaining the solution quality and consistency of the latter.

VI. Conclusion, Limitations, and Future Work

We presented MGS, a multi-graph search framework for motion planning that departs from the traditional unidirectional and bidirectional search paradigm. By maintaining multiple subgraphs anchored at strategically chosen root configurations, MGS focuses exploration on high-potential regions in the state space. MGS retains the completeness and bounded-suboptimality guarantees of classical search-based planners, while our experiments demonstrate that it achieves substantial improvements in planning efficiency. While effective, the current approach has limitations that point to exciting future directions. The root selection strategy reasons in end-effector workspace, implicitly assuming that the end-effector position is a sufficient proxy for identifying important regions in configuration-space, and tends to produce roots that are close to obstacle boundaries. For tasks with different constraints—such as preferring greater clearance—the workspace BFS may miss critical regions. Since MGS is agnostic to the root selection strategy, these limitations can be addressed by plugging in alternative strategies—such as curating libraries of kinematically favorable configurations, reasoning about the

full robot body, or learning root placement policies from data. Dynamically selecting which subgraphs to expand during search is another natural extension. More broadly, integrating learned motion primitives or manipulation skills as subgraph components could extend MGS beyond collision-free planning to contact-rich tasks, and exploiting the multi-graph structure for parallelization could yield further speedups.

REFERENCES

[1] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Rob. & Aut. Mag.*, vol. 19, no. 4, pp. 72–82, 2012.

[2] I. Mishani, Y. Shaoul, R. Natarajan, J. Li, and M. Likhachev, "Srmp: Search-based robot motion planning library," 2025.

[3] S. LAVALLE, "Rapidly-exploring random trees : a new tool for path planning," *Research Report 9811*, 1998.

[4] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[5] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," 2011.

[6] B. Hayes-Roth and F. Hayes-Roth, "A cognitive model of planning," *Cognitive science*, vol. 3, no. 4, pp. 275–310, 1979.

[7] A. Morris, J. Phillips, K. Huang, and F. Cushman, "Generating options and choosing between them depend on distinct forms of value representation," *Psychological Science*, vol. 32, no. 11, pp. 1731–1746, 2021.

[8] C. Celemin, R. Pérez-Dattari, E. Chisari, G. Franzese, L. de Souza Rosa, R. Prakash, Z. Ajanović, M. Ferraz, A. Valada, J. Kober *et al.*, "Interactive imitation learning in robotics: A survey," *Foundations and Trends® in Robotics*, vol. 10, no. 1-2, pp. 1–197, 2022.

[9] A. Orthey, C. Chamzas, and L. E. Kavraki, "Sampling-based motion planning: A comparative review," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 7, 2023.

[10] L. Nardi and C. Stachniss, "Experience-based path planning for mobile robots exploiting user preferences," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 1170–1176.

[11] T. Marcucci, M. Halm, W. Yang, D. Lee, and A. D. Marchese, "A biconvex method for minimum-time motion planning through sequences of convex sets," *arXiv preprint arXiv:2504.18978*, 2025.

[12] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[13] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime search in dynamic graphs," *Artificial Intelligence*, vol. 172, no. 14, pp. 1613–1643, 2008.

[14] S. Koenig and M. Likhachev, "D*lite," in *Eighteenth National Conference on Artificial Intelligence*. USA: American Association for Artificial Intelligence, 2002, p. 476–483.

[15] ——, "Fast replanning for navigation in unknown terrain," *IEEE transactions on robotics*, vol. 21, no. 3, pp. 354–363, 2005.

[16] M. Likhachev and D. Ferguson, "Planning long dynamically feasible maneuvers for autonomous vehicles," *The International Journal of Robotics Research*, vol. 28, no. 8, pp. 933–945, 2009.

[17] B. J. Cohen, S. Chitta, and M. Likhachev, "Search-based planning for manipulation with motion primitives," in *IEEE International Conference on Robotics and Automation*, 2010, pp. 2902–2908.

[18] B. J. Cohen, G. Subramania, S. Chitta, and M. Likhachev, "Planning for manipulation with adaptive motion primitives," in *IEEE International Conference on Robotics and Automation*, 2011, pp. 5478–5485.

[19] M. Phillips, B. Cohen, S. Chitta, and M. Likhachev, "E-graphs: Bootstrapping planning with experience graphs," in *Proceedings of Robotics: Science and Systems*, Sydney, Australia, July 2012.

[20] M. Phillips, V. Narayanan, S. Aine, and M. Likhachev, "Efficient search with an ensemble of heuristics." in *IJCAI*, 2015, pp. 784–791.

[21] S. Aine, S. Swaminathan, V. Narayanan, V. Hwang, and M. Likhachev, "Multi-heuristic a*," *IJRR*, 2016.

[22] S. Chitta, B. Cohen, and M. Likhachev, "Planning for autonomous door opening with a mobile manipulator," in *2010 IEEE International Conference on Robotics and Automation*. IEEE, 2010, pp. 1799–1806.

[23] Y. Shaoul, I. Mishani, M. Likhachev, and J. Li, "Accelerating search-based planning for multi-robot manipulation by leveraging online-generated experiences," in *International Conference on Automated Planning and Scheduling*, 2024.

[24] Y. Shaoul, R. Veerapaneni, M. Likhachev, and J. Li, "Unconstraining multi-robot manipulation: Enabling arbitrary constraints in ecbs with bounded sub-optimality," in *International Symposium on Combinatorial Search*, 2024.

[25] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *IEEE international conference on robotics and automation*, 2009.

[26] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 4569–4574.

[27] M. Mukadam, J. Dong, X. Yan, F. Dellaert, and B. Boots, "Continuous-time gaussian process motion planning via probabilistic inference," *The International Journal of Robotics Research*, 2018.

[28] J. Schulman, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, and

P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization." in *Robotics: science and systems*, vol. 9, no. 1. Berlin, Germany, 2013, pp. 1–10.

[29] A. H. Qureshi, A. Simeonov, M. J. Bency, and M. C. Yip, "Motion planning networks," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 2118–2124.

[30] M. Dalal, J. Yang, R. Mendonca, Y. Khaky, R. Salakhutdinov, and D. Pathak, "Neural mp: A generalist neural motion planner," *arXiv preprint arXiv:2409.05864*, 2024.

[31] J. Carvalho, A. T. Le, M. Baierl, D. Koert, and J. Peters, "Motion planning diffusion: Learning and planning of robot motions with diffusion models," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023.

[32] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7087–7094.

[33] J. Kuffner and S. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2. IEEE, 2000, pp. 995–1001.

[34] Z. Sun, J. Wang, and M. Q.-H. Meng, "Multi-tree guided efficient robot motion planning," *Procedia Computer Science*, vol. 209, pp. 31–39, 2022, proceedings of the 2022 International Symposium on Biomimetic Intelligence and Robotics (ISBIR).

[35] I. Pohl, "Bi-directional and heuristic search in path problems," Ph.D. dissertation, Department of Computer Science, Stanford University, 1969.

[36] H. Kaindl and G. Kainz, "Bidirectional heuristic search reconsidered," *Journal of Artificial Intelligence Research*, vol. 7, pp. 283–317, 1997.

[37] F. Islam, V. Narayanan, and M. Likhachev, "A*-connect: Bounded suboptimal bidirectional heuristic search," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 2752–2758.

[38] S. Lavasani, L. Siag, S. S. Shperberg, A. Felner, and N. R. Sturtevant, "Anchor search: A unified framework for suboptimal bidirectional search," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, no. 25, 2025, pp. 27 045–27 053.

[39] S. Richter, M. Helmert, and M. Westphal, "Landmarks revisited." in *AAAI*, vol. 8, 2008, pp. 975–982.

[40] S. Richter and M. Westphal, "The lama planner: Guiding cost-based anytime planning with landmarks," *Journal of Artificial Intelligence Research*, vol. 39, pp. 127–177, 2010.

[41] F. Pommerening and M. Helmert, "Incremental lm-cut," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 23, 2013, pp. 162–170.

[42] J. Segovia-Aguas, S. J. Celorrio, L. Sebastiá, and A. Jonsson, "Scaling-up generalized planning as heuristic search with landmarks," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 15, no. 1, 2022, pp. 171–179.

[43] C. Büchner, T. Keller, S. Eriksson, and M. Helmert, "Landmark progression in heuristic search," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 33, no. 1, pp. 70–79, Jul. 2023.

[44] S. Goel and M. Huber, "Subgoal discovery for hierarchical reinforcement learning using learned policies," in *FLAIRS*, 2003, pp. 346–350.

[45] B. Eysenbach, R. R. Salakhutdinov, and S. Levine, "Search on the replay buffer: Bridging planning and reinforcement learning," *Advances in neural information processing systems*, vol. 32, 2019.

[46] A. Bagaria, J. K. Senthil, and G. Konidaris, "Skill discovery for exploration and planning using deep skill graphs," in *International conference on machine learning*. PMLR, 2021, pp. 521–531.

[47] F. Islam, O. Salzman, and M. Likhachev, "Provable indefinite-horizon real-time planning for repetitive tasks," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 29, no. 1, pp. 716–724, May 2021.

[48] I. Mishani, H. Feddock, and M. Likhachev, "Constant-time motion planning with anytime refinement for manipulation," 2023.

[49] T. Marcucci, M. Petersen, D. von Wrangel, and R. Tedrake, "Motion planning around obstacles with convex optimization," 2022.

[50] L. Cohen, M. Greco, H. Ma, C. Hernández, A. Felner, T. S. Kumar, and S. Koenig, "Anytime focal search with applications." in *IJCAI*, 2018, pp. 1434–1441.

[51] A. Zou, M. S. Saleem, and M. Likhachev, "Attractor-based closed list search: Sparsifying the closed list for efficient memory-constrained planning," in *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, 2025, pp. 9031–9038.

[52] D. De Champeaux, "Bidirectional heuristic search again," *Journal of the ACM (JACM)*, vol. 30, no. 1, pp. 22–32, 1983.

[53] C. Chamzas, C. Quintero-Pena, Z. Kingston, A. Orthey, D. Rakita, M. Gleicher, M. Toussaint, and L. E. Kavraki, "Motionbenchmaker: A tool to generate and benchmark motion planning datasets," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 882–889, 2021.

[54] D. Coleman, I. A. Sucan, S. Chitta, and N. Correll, "Reducing the barrier to entry of complex robotic software: a moveit! case study," *CoRR*, vol. abs/1404.3785, 2014.

[55] J. Pan, S. Chitta, and D. Manocha, "Fcl: A general purpose library for collision and proximity queries," in *IEEE International Conference on Robotics and Automation*, 2012.

[56] D. Devaurs, T. Siméon, and J. Cortés, "Enhancing the transition-based rrt to deal with complex cost spaces,"

in *2013 IEEE International Conference on Robotics and Automation*, 2013.

[57] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," in *Proceedings of international conference on robotics and automation*, 1997.

[58] Z. Sun, J. Wang, and M. Q.-H. Meng, "Multi-tree guided efficient robot motion planning," *Procedia Computer Science*, vol. 209, pp. 31–39, 2022.

[59] M. Phillips, M. Likhachev, and S. Koenig, "Pa*se: Parallel a* for slow expansions," in *Proceedings of the International Conference on Automated Planning and Scheduling*, 2014.

APPENDIX

We first give additional algorithmic details, including pseudocode, then discuss attractors and their geometric role in guiding multi-graph search, and finally summarize implementation details. We also report additional experimental results: per-scenario breakdowns of Section V and ablations on the time-limit and the maximum number of sub-graphs parameters. Finally, we present an additional domain demonstrating the generalization of MGS, discuss the impact of joint limits relative to baselines, and analyze failure cases.

A. Algorithmic Details

We present pseudocode for MERGESUBGRAPHS (Algorithm 3) and summarize TRYTOCONNECT, GETCONNECTINGPATH, and CHOOSEMERGINGORDER below.

TRYTOCONNECT checks whether a state q, expanded in a sub-graph, can be connected to states that already belong to other sub-graphs. Any method that solves the two-point boundary value problem between q and states in the target sub-graphs can be used. Here we use linear interpolation in configuration space: for each nearest neighbor q' on the frontier of a target sub-graph G_j, the interpolated path from q to q' is collision-checked. If the path is collision-free, the connection (q', G_j) is recorded. GETCONNECTINGPATH returns the corresponding collision-free interpolated path.

MERGESUBGRAPHS (Algorithm 3) transfers all states and edges from G_{to} into G_{from}. Starting from the merge point q_{merge}, a BFS traverses G_{to} using the stored edges and their costs (Line 4). For each state s' adjacent to the current state s, the g-value is computed using the g-value of s in G_{from} plus the edge cost. Each edge is transferred to G_{from}, and each newly visited state is added via ADDSTATETO (Line 14) if not already present (Line 15). When merging into the anchor G_1, all states are inserted into OPEN to allow re-expansion under the anchor's heuristic ordering. When merging two connect searches, closed states remain closed, avoiding redundant expansions until the eventual merge into the anchor.

CHOOSEMERGINGORDER(G_i, G_j, $\mathcal{G}$) determines which sub-graph absorbs the other during a connect-connect merge. If either sub-graph is the anchor G_1, the anchor always receives. Otherwise, the sub-graph whose root has a smaller admissible heuristic estimate $h(r, q_{\text{start}})$—estimated to be closer to the start—is designated as G_{from}, as it is more likely to merge into the anchor sooner.

Algorithm 3: MERGESUBGRAPHS

INPUT: Receiving sub-graph G_{from}, merged sub-graph G_{to}, merge point $q_{\text{merge}} \in V_{\text{to}}$, sub-graph collection $\mathcal{G}$
OUTPUT: Updated collection $\mathcal{G}$

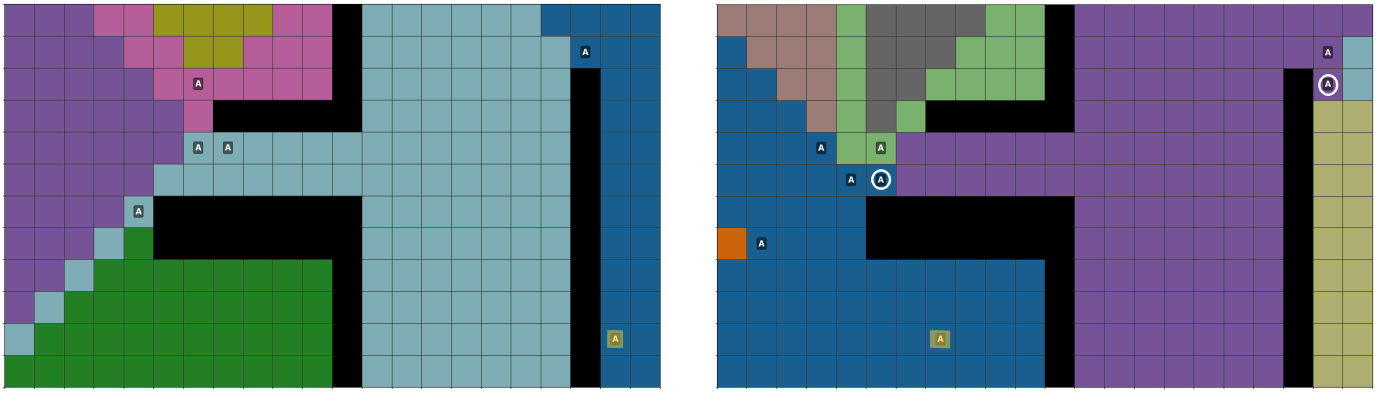
```
   // BFS from merge point: propagate g-values and
      transfer edges/states
1  Q ← {q_merge} visited ← {q_merge}
2  while Q ≠ ∅ do
3      s ← Q.dequeue()
4      foreach edge (s, s', c(s, s')) from s in G_to do
5          if s' ∉ visited then
6              g(s') ← G_from.getGvalue(s) + c(s, s') ;
                 // Propagate from G_to's g-values
7              f(s') ← g(s') + h(s')
8              visited ← visited ∪ {s'}
9              AddStateTo(G_from, s')
10             Q.enqueue(s')
11         G_from.AddEdge((s, s', c(s, s')))
12 G ← G \ {G_to}
13 return G

14 Function AddStateTo(G, s):
15     if s ∈ V_G then
16         return
17     if G = G_1 then
           // Anchor merge: state enters OPEN for
              re-expansion
18         G.OPEN().Insert(s)
19         if f(s) ≤ ε · f_min then
20             G.FOCAL().Insert(s)
21     else
           // Connect-connect merge: preserve closed
              status
22         if s ∈ G_to.CLOSED() then
23             G.CLOSED().Insert(s)
24         else
25             G.OPEN().Insert(s)
```

B. Backward and Forward Attractors

The root selection procedure (Algorithm 2) identifies attractor states in the end-effector workspace through backward BFS from the goal position. Here we provide further intuition on the geometric role of these attractors and the distinction between backward and forward attractors. As the BFS wavefront propagates outward from the goal, it encounters obstacles and flows around them. An attractor emerges at a location w where the greedy path from w back toward the goal must diverge due to an obstacle: the greedy predecessor of w's neighbor differs from the BFS parent (Algorithm 2, Line 21), indicating that the obstacle geometry forces a detour. Geometrically, backward attractors tend to form on the side of obstacles *facing away from the goal*, effectively "hugging" the obstacle from the side opposite the BFS root. Each attractor thus marks the entrance to a region that is not directly reachable from the goal without navigating around an obstacle, making it a natural candidate for a search root that can explore that region locally.

While backward attractors provide broad coverage of obstacle boundaries, they may be hard to connect to in full configuration space. To ensure coverage along the actual start-to-goal corridor, Algorithm 2 traces the BFS policy *forward* from the start end-effector position (Line 30), collecting the attractors encountered along this path. This forward rollout ensures obstacle sides are represented with respect to *both* the



(a) Backward attractors from the goal.

(b) Forward attractors from the start.

Fig. 6: Comparison of backward and forward attractor discovery. Backward (left) attractors are often generated facing away from the goal, while forward (right) attractors face toward the goal. The attractors discovered by the forward policy rollout (not the full BFS) are marked with a white circle. The root of the search is highlighted in yellow.

start and the goal, yielding roots that are both broadly placed near critical boundaries and concentrated along the relevant corridor (Fig. 6).

C. Implementation Details

We describe additional implementation choices not fully specified in the main text.

Motion Primitives and Discretization. The implicit graph is constructed using single-joint motion primitives: for each joint $j \in \{1, \ldots, d\}$, the primitive applies a displacement of $\pm\Delta\theta_j$ to joint j while holding all other joints fixed. We use adaptive motion primitive—long and short—where if the current state is within a certain distance threshold of the goal or start, we use both short and long primitives, otherwise only long primitives. All edges have uniform cost, consistent with the search-based baselines described in Section V-C. The joint resolution $\Delta\theta_j$ may vary per joint—typically finer for wrist joints and coarser for shoulder or base joints—and is set to match the SRMP framework [2] used in our experiments with a resolution of 1 degree for revolute joints and 1 cm for prismatic joints. Please note that while we discretize for planning, the resulting solutions are in continuous space and the goal reached is accurate (the state datastructure contains both the discretized configuration and the underlying continuous configuration).

Root Selection. The end-effector workspace is discretized into a 3D occupancy grid, where each voxel is marked as occupied if it overlaps with any obstacle geometry. We use a 2 cm voxel size for manipulation and a 5 cm voxel size for mobile manipulation, and inflate obstacles by 5 cm based on the gripper geometry. This provides conservative clearance during workspace reasoning: the signed distance field (SDF) derived from the occupancy grid is inflated around the end-effector, ensuring that identified attractors maintain a margin from obstacle surfaces. For mapping workspace attractors to configuration space, we use differential inverse kinematics seeded with the configuration of the previously mapped attractor (starting from the start configuration). If the IK solver fails to converge or returns a configuration in collision, the attractor is discarded. When the number of attractors exceeds the sub-graph budget m, k-means clustering is applied in workspace coordinates, and the attractor closest to each cluster centroid is selected as the representative root.

Sub-graph Connections. For TRYTOCONNECT, we use a single nearest neighbor ($k = 1$) on the frontier of each target sub-graph when attempting connections.

D. Per-Scenario Experimental Results

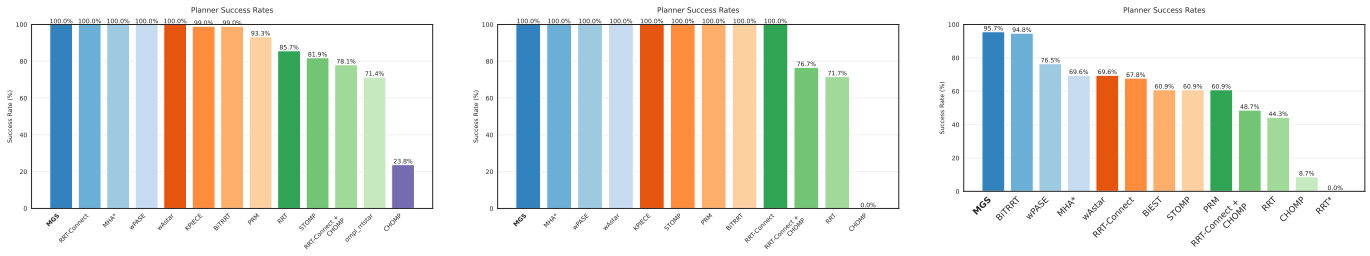
We provide detailed per-scenario breakdowns of the experimental results summarized in Section V. We show success rates for each scenario (Fig. 7) and pairwise cost comparison matrices (Figs. 11 and 10) for manipulation and mobile manipulation scenarios, respectively.

E. Time Limit Ablation

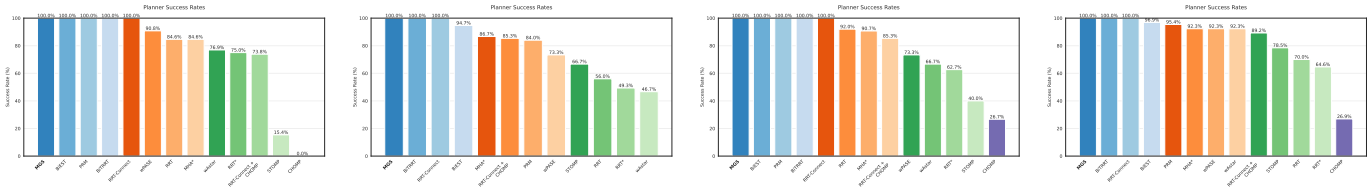
We study the impact of the planning time limit on performance. Fig. 8 shows success rates and solution costs for varying time limits (5s, 10s, 20s) in the shelf pick-and-place scenario and the cage extraction scenario. The relative performance between planners remains consistent across time limits. Notably, sampling-based planners occasionally exhibit decreased success rates at larger time limits (e.g., 10s to 20s); this counterintuitive behavior stems from statistical variance inherent to their randomized nature, unlike search-based planners which are deterministic and show monotonic improvement with increased time.

F. Computational Overhead vs. Number of Sub-graphs

We analyze how the computational overhead of MGS scales with the number of sub-graphs. Fig. 9 shows the ratio of time spent in auxiliary operations—including TRYTOCONNECT, MERGESUBGRAPHS, and sub-graph bookkeeping—to the time spent in state expansion. The overhead exhibits an approximately linear trend with respect to the number of sub-graphs: each additional sub-graph introduces connection

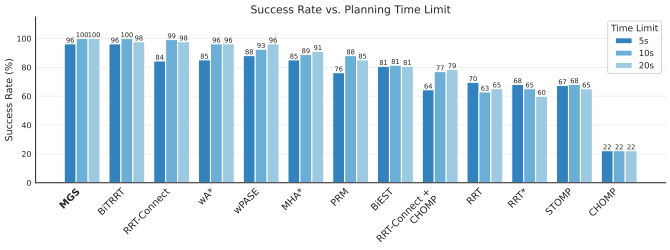


(a) Shelf pick-and-place (manip.). (b) Bin picking (manip.). (c) Cage extraction (manip.).

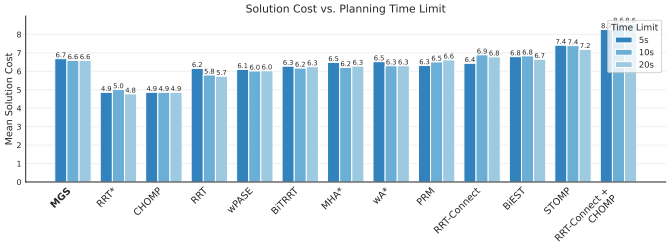


(d) Low-clearance (mobile). (e) Deep shelf (mobile). (f) Cluttered table (mobile). (g) Warehouse (mobile).

Fig. 7: Success rates for all scenarios. Manipulation scenarios (a–c) use a 7-DOF Franka Panda; mobile manipulation scenarios (d–g) use a 9-DOF Ridgeback + UR10e. Each bar represents the percentage of queries solved within the 5-second time limit.



(a) Success rates for varying time limits.



(b) Solution costs for varying time limits.

Fig. 8: Ablation on the planning time limit for time limits of 5s, 10s, and 20s.

attempts and potential merge operations that scale with the number of existing sub-graphs.

Empirically, we found that $m = 10$ sub-graphs provides a good trade-off between performance gains and computational cost. Beyond this point, the improvements in success rate and solution quality begin to stagnate, while the overhead continues to grow linearly. This observation guided our choice of the default sub-graph budget used throughout the experiments.

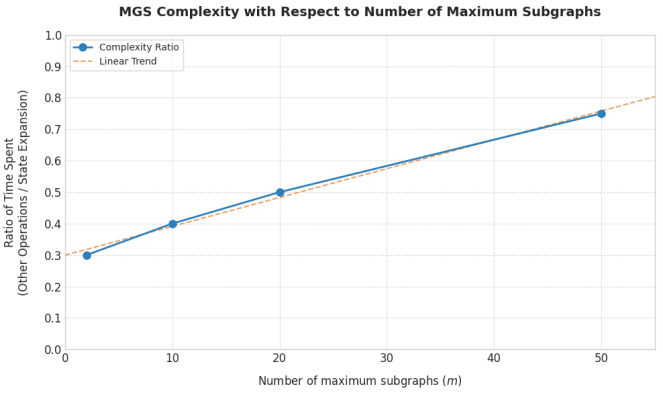
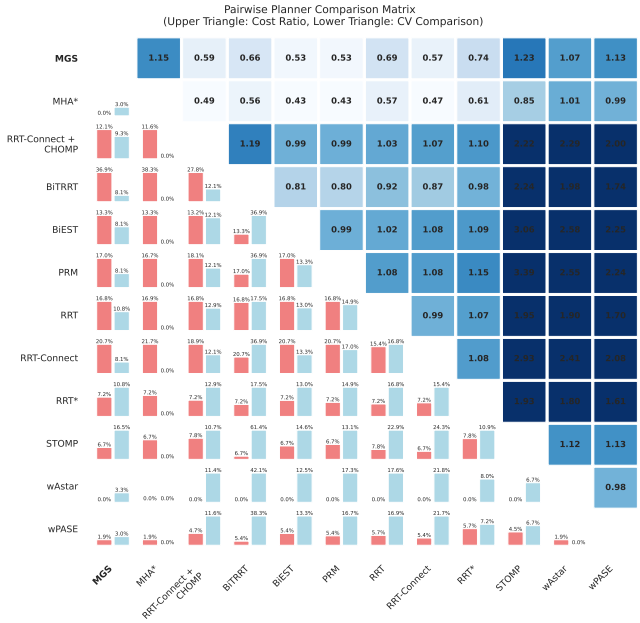


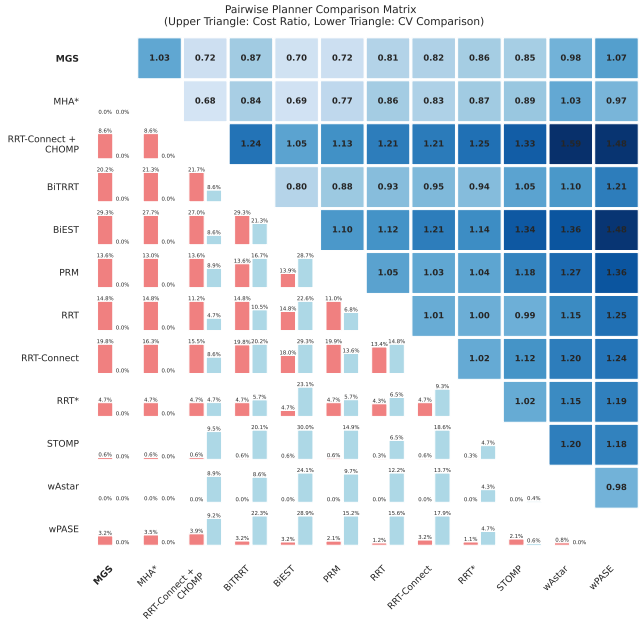
Fig. 9: Computational overhead as a function of the number of sub-graphs. The y-axis shows the ratio of time spent in auxiliary operations (TRYTOCONNECT, MERGESUBGRAPHS, subgraph management) to time spent in state expansion. As the number of sub-graphs increases, the overhead grows approximately linearly.

G. Generalization

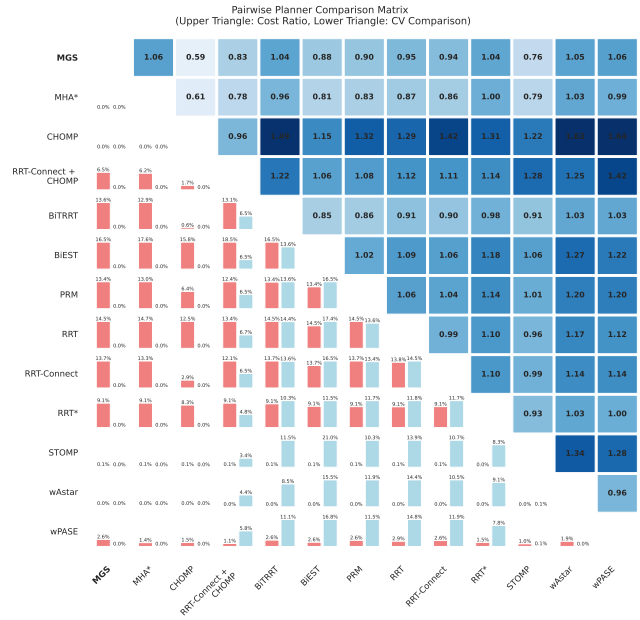
MGS is designed as a general framework for planning with undirected graphs. The multi-directional approach is a general planning paradigm, complemented by a root selection procedure that can be adapted to different domains. To demonstrate this generalization, we applied MGS to 3D navigation problems (x-y-θ) with a footprint (e.g., planning for the base of a mobile manipulator requires full-body collision checking). We constructed occupancy grids of the environment, computed attractors in the 2D workspace (x-y) as in Section IV-D, and mapped them to (x-y-θ) configurations using a simple heuristic: for attractors closer to the start, we set θ to match the start orientation; for attractors closer to the goal, we set θ



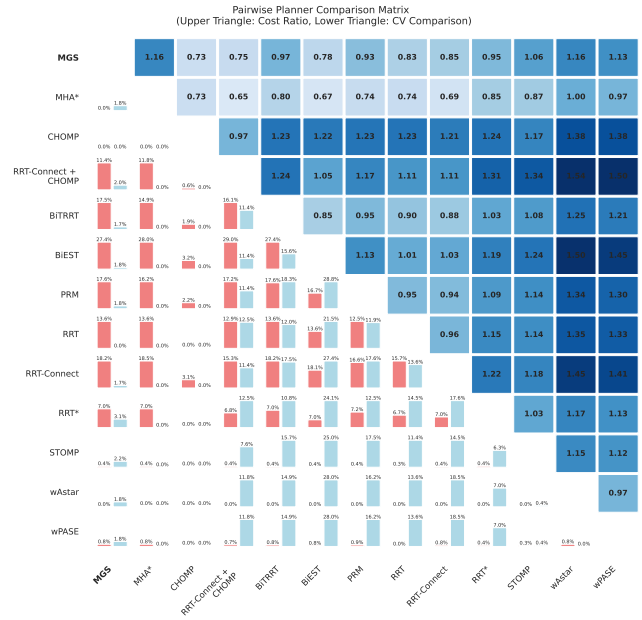
(a) Low-clearance passage.



(b) Deep shelf reach.

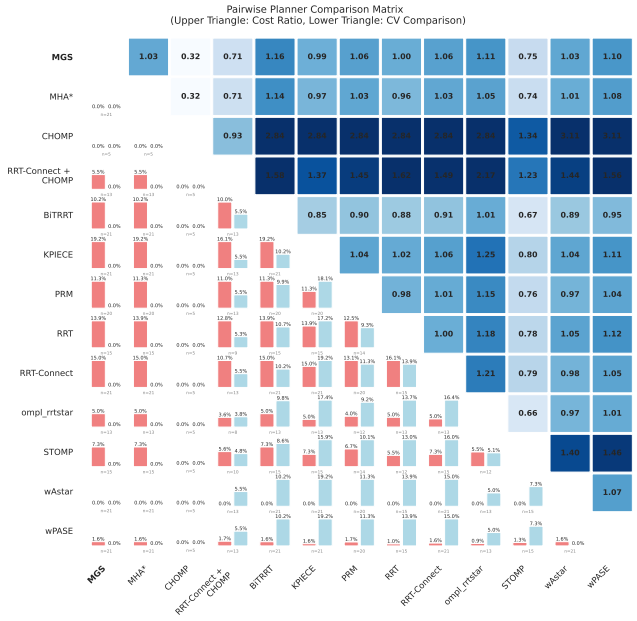


(c) Cluttered table.

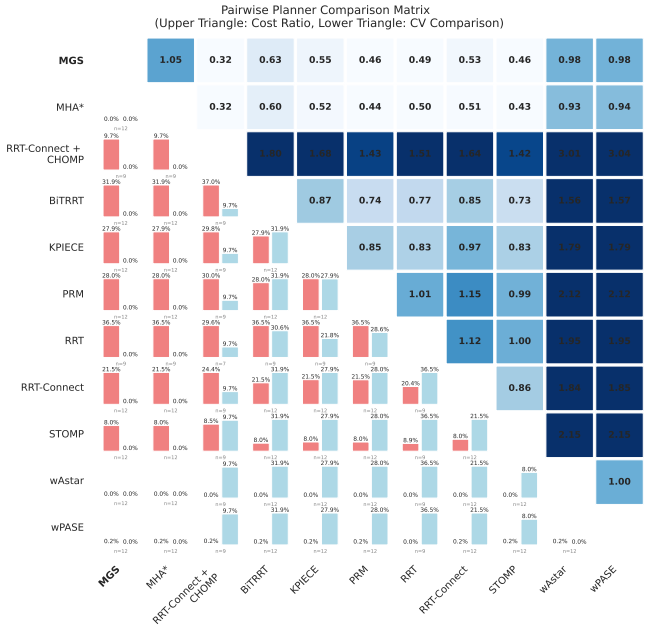


(d) Combined warehouse.

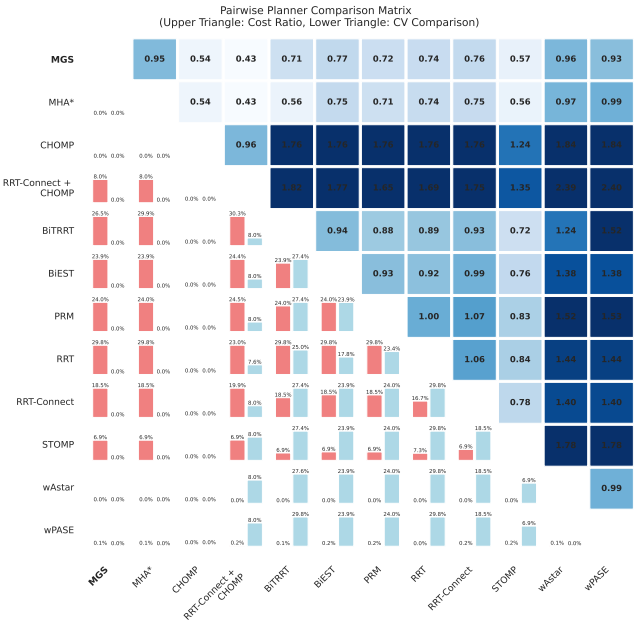
Fig. 10: Pairwise comparison matrices for mobile manipulation scenarios (9-DOF Ridgeback + UR10e). The upper triangle shows the pairwise relative cost ratio (row planner divided by column planner), computed only on queries where both planners succeeded—values below 1.0 indicate the row planner produces lower-cost solutions. The lower triangle shows the coefficient of variation (CV) of these cost ratios, measuring consistency—lower values indicate more predictable pairwise behavior.



(a) Shelf pick-and-place.



(b) Bin picking.



(c) Cage extraction.

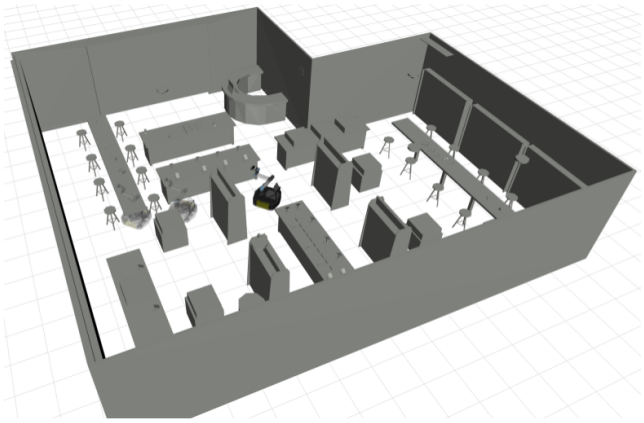
Fig. 11: Pairwise comparison matrices for manipulation scenarios (7-DOF Franka Panda). As in Fig. 10, the upper triangle shows pairwise relative cost ratios and the lower triangle shows consistency (CV). Results exhibit similar trends: MGS produces competitive solution costs with low variance across queries.

to match the goal orientation. Collision checking is performed for the entire robot—mobile base and arm. See Fig. 12 for an example environment, its occupancy grid representation, and a path planned by MGS.

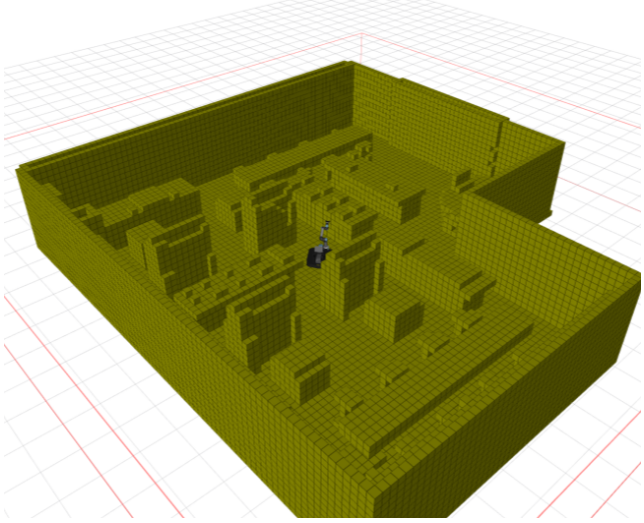
H. Impact of Joint Limits

Search-based methods, including MGS, explore states systematically via successor generation from the current state. As a result, performance is largely unaffected by the range of joint limits: increasing the limits of a particular joint (e.g., the base joints of a mobile manipulator) does not alter the search behavior or solution quality, provided the start and goal configurations remain reachable.

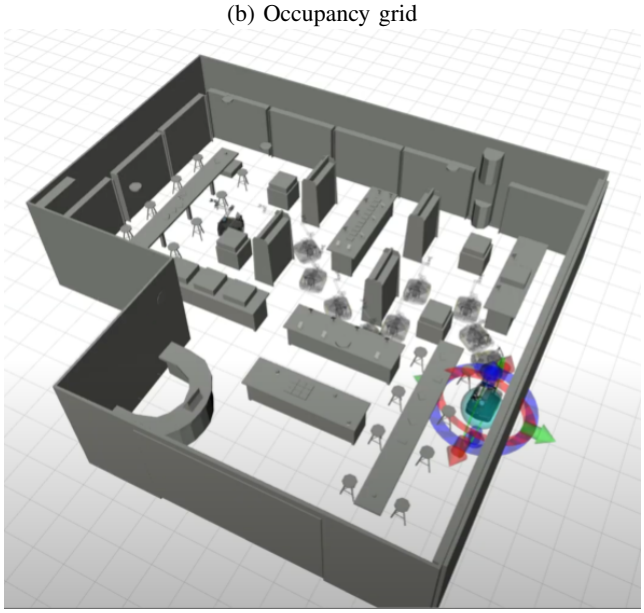
Sampling-based planners, in contrast, are highly sensitive to joint-limit ranges. Wider joint limits enlarge the sampling domain, diluting the probability of sampling configurations in task-relevant regions. This leads to degraded planning performances including success rate and planning time, and the solutions are often highly suboptimal. The effect is particularly



(a) AWS RobotMaker Bookstore World Environment https://github.com/aws-robotics/aws-robomaker-bookstore-world



(b) Occupancy grid



(c) Path planned by MGS

Fig. 12: Example 3D navigation environment (top) and its corresponding occupancy grid representation (middle). The occupancy grid is used for attractor computation in the root selection procedure. The bottom image shows a path planned by MGS in this environment, demonstrating the framework's applicability beyond manipulation tasks.

pronounced in mobile manipulation, where the base joints can have large ranges relative to the arm joints, skewing the sampling distribution away from the manipulation workspace.

I. Failure Cases

While MGS improves performance in many scenarios, several limitations exist. The root selection procedure computes attractors via BFS in the workspace, which may not reflect configuration-space connectivity; if an obstacle blocks the arm but not the end-effector path, no attractor will be placed to help navigate around it. Similarly, if a critical narrow passage in configuration space does not coincide with any attractor, MGS gains no advantage over standard bidirectional search. Mapping workspace attractors to configuration-space roots relies on differential IK, which may fail in highly constrained scenarios or near singularities, causing some attractors to be discarded. Additionally, when paths are relatively direct, maintaining multiple sub-graphs introduces overhead without proportional benefit, and a well-guided unidirectional search may outperform MGS.

Finally, because attractors are identified where the BFS wavefront diverges around obstacles, they inherently "hug" obstacle boundaries, and paths through these intermediate goals tend to remain close to obstacles. In applications where maximizing clearance is desirable—such as safety-critical tasks or environments with uncertain obstacle geometry—this behavior may be undesirable, and alternative attractor selection strategies that balance efficiency with clearance could address this limitation.