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.

Algorithm 3: MERGESUBGRAPHS

```
INPUT: Receiving sub-graph $G_{\text{from}}$, merged sub-graph $G_{\text{to}}$, 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  $Q \leftarrow \{q_{\text{merge}}\}$ visited $\leftarrow \{q_{\text{merge}}\}$
2  while $Q \neq \emptyset$ do
3      $s \leftarrow Q$.dequeue()
4      foreach edge $(s, s', c(s, s'))$ from $s$ in $G_{to}$ do
5          if $s' \notin$ visited then
6              $g(s') \leftarrow G_{from}.getGvalue(s) + c(s, s')$ ;
                 // Propagate from $G_{\text{to}}$'s g-values
7              $f(s') \leftarrow g(s') + h(s')$
8              visited $\leftarrow$ visited $\cup \{s'\}$
9              ADDSTATETO($G_{\text{from}}$, $s'$)
10             $Q$.enqueue($s'$)
11         $G_{\text{from}}$.AddEdge($(s, s', c(s, s'))$)
12 $\mathcal{G} \leftarrow \mathcal{G} \setminus \{G_{\text{to}}\}$
13 return $\mathcal{G}$
14 Function ADDSTATETO($G$, $s$):
15     if $s \in V_G$ then
16         return
17     if $G = G_1$ then
           // Anchor merge: state enters OPEN for
              re-expansion
18         $G$.OPEN().Insert($s$)
19         if $f(s) \leq \epsilon \cdot f_{\min}$ then
20             $G$.FOCAL().Insert($s$)
21     else
           // Connect-connect merge: preserve closed
              status
22         if $s \in G_{to}$.CLOSED() then
23             $G$.CLOSED().Insert($s$)
24         else
25             $G$.OPEN().Insert($s$)
```

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.

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 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.

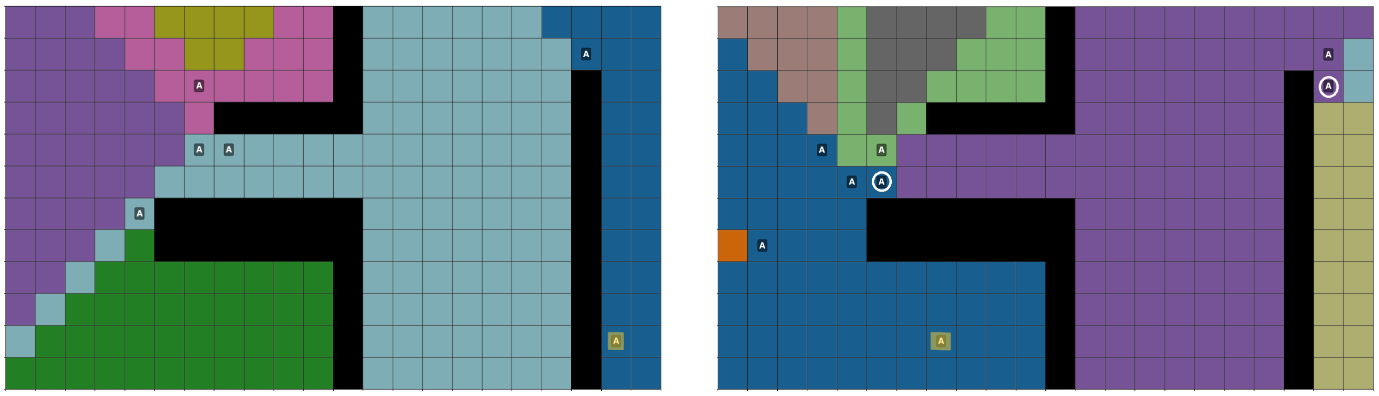

(a) Backward attractors from the goal.
(b) Forward attractors from the start.

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.

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.

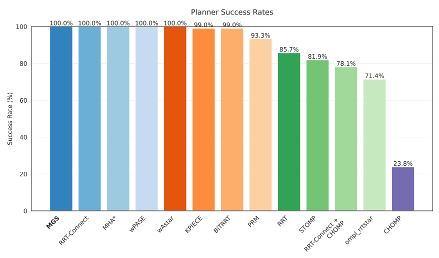
F. Computational Overhead vs. Number of Sub-graphs

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

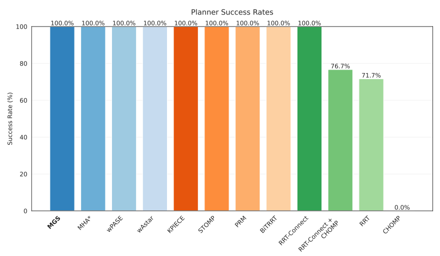
Empirically, we found that $m = 10$ sub-graphs provides a good trade-off between performance gains and computational cost. Beyond this point, the improvements in success rate and solution quality begin to stagnate, while the overhead continues to grow linearly. This observation guided our choice of the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

G. Generalization

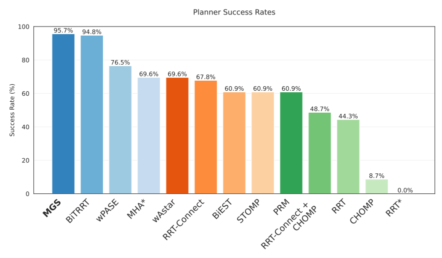
MGS is designed as a general framework for planning with undirected graphs. The multi-directional approach is a general planning paradigm, complemented by a root selection procedure that can be adapted to different domains. To demonstrate this generalization, we applied MGS to 3D navigation



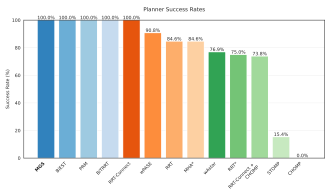
(a) Shelf pick-and-place (manip.).



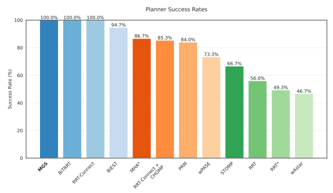
(b) Bin picking (manip.).



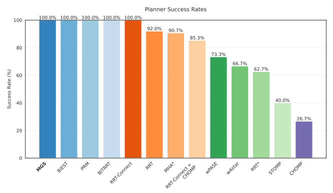
(c) Cage extraction (manip.).



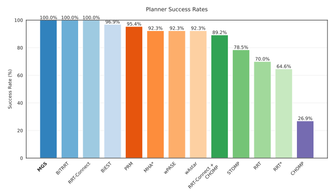
(d) Low-clearance (mobile).



(e) Deep shelf (mobile).

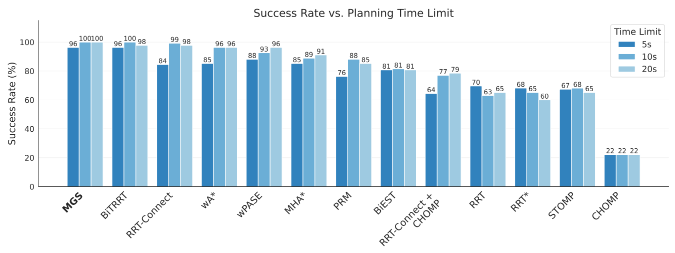


(f) Cluttered table (mobile).

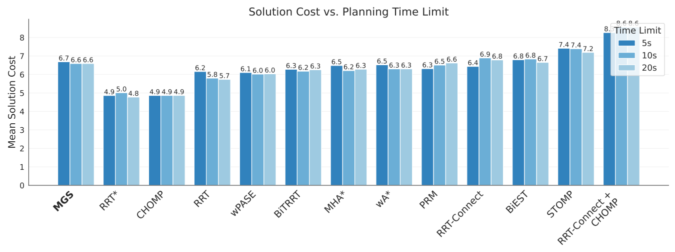


(g) Warehouse (mobile).

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.

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 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

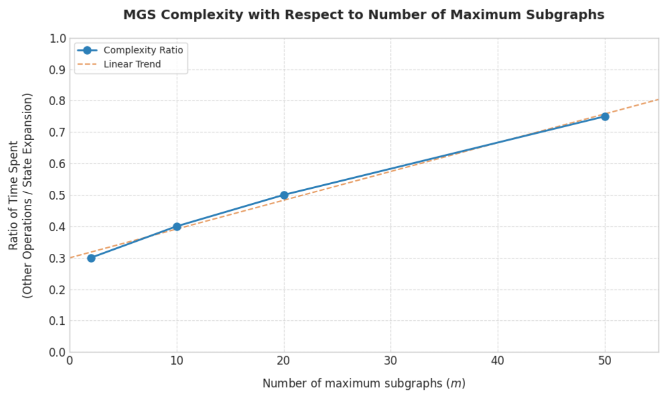


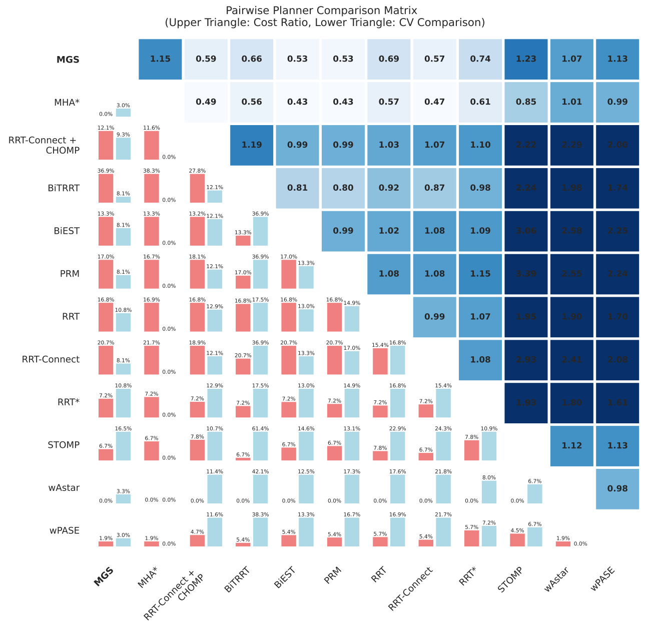
Fig. 9: Computational overhead as a function of the number of sub-graphs. The y-axis shows the ratio of time spent in auxiliary operations (TRYTOCONNECT, MERGESUBGRAPHS, sub-graph management) to time spent in state expansion. As the number of sub-graphs increases, the overhead grows approximately linearly.

a path planned by MGS.

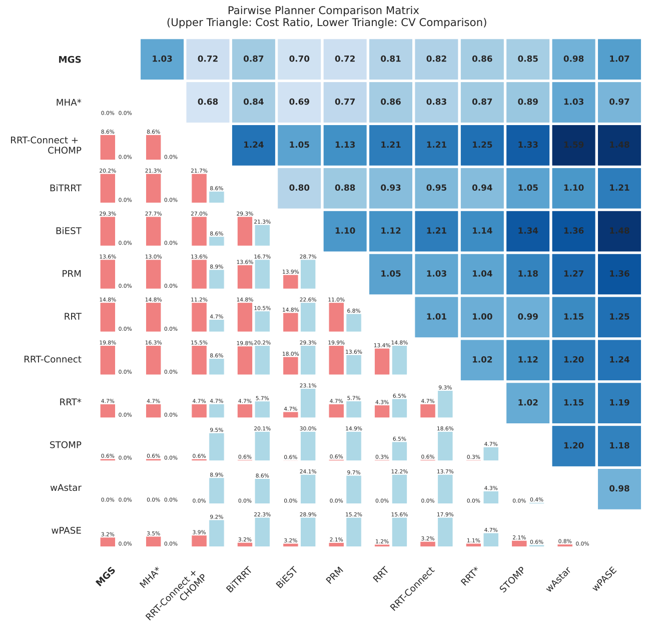
H. Impact of Joint Limits

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.

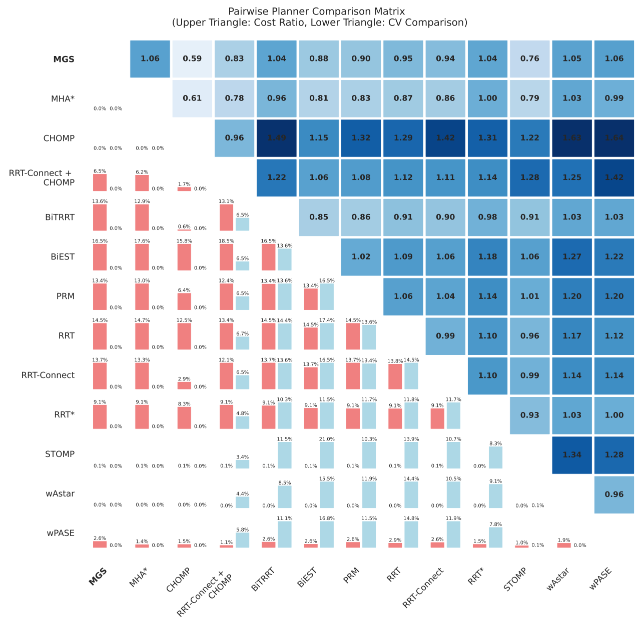
Sampling-based planners, in contrast, are highly sensitive to joint-limit ranges. Wider joint limits enlarge the sampling domain, diluting the probability of sampling configurations



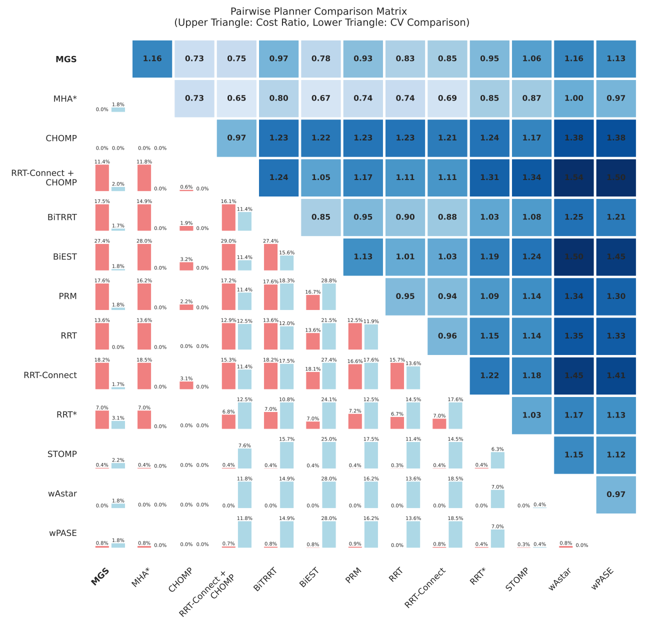
(a) Low-clearance passage.



(b) Deep shelf reach.

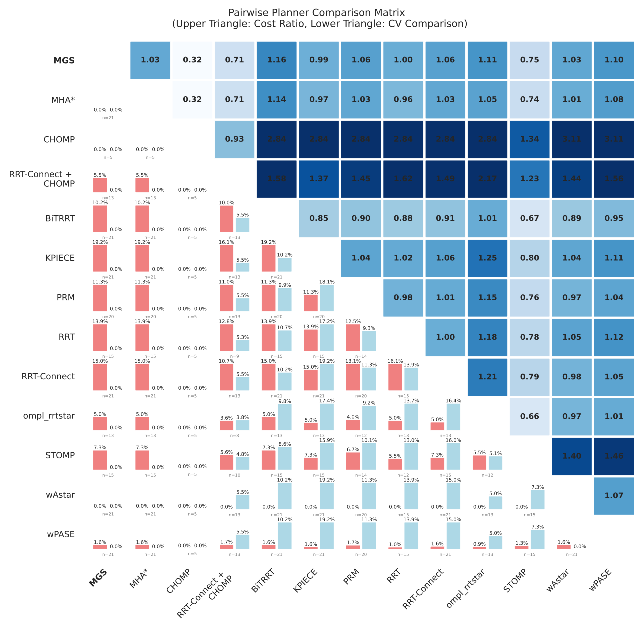


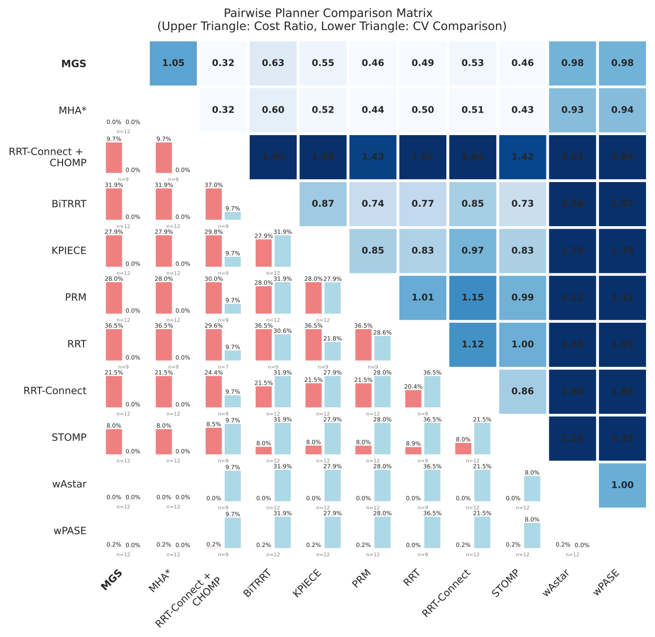
(c) Cluttered table.



(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.

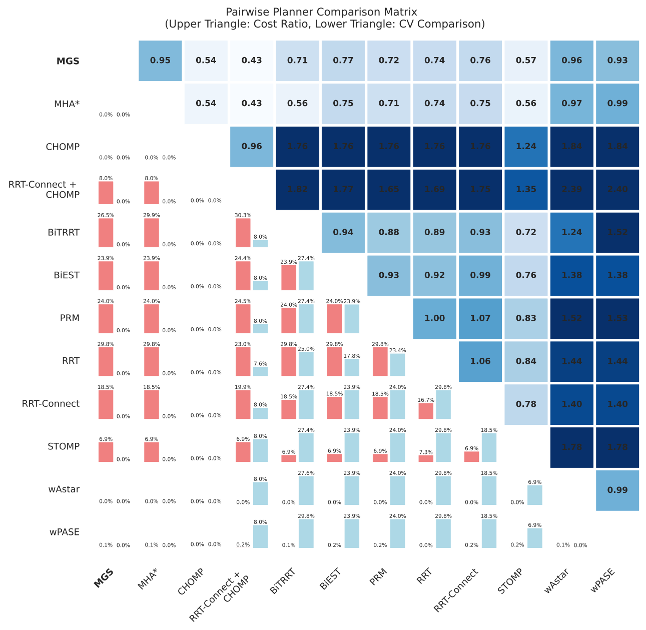
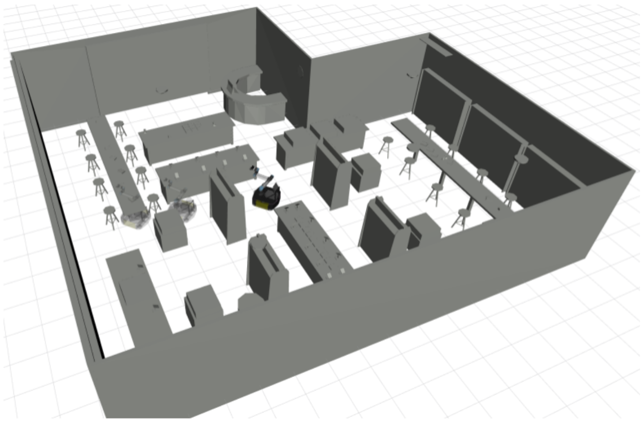

(c) Cage extraction.

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.

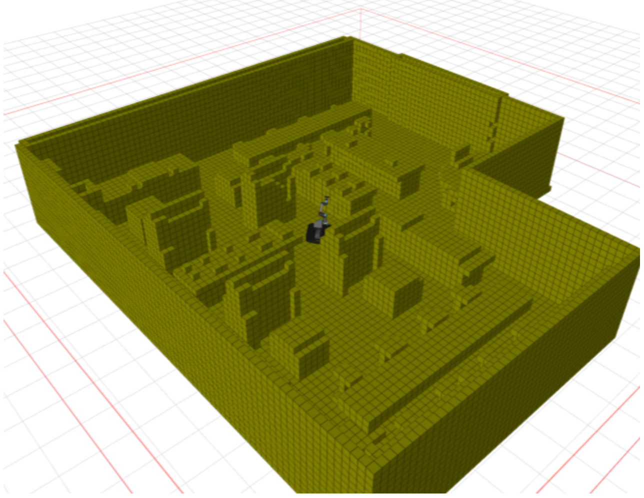
in task-relevant regions. This leads to degraded planning performances including success rate and planning time, and the solutions are often highly suboptimal. The effect is particularly pronounced in mobile manipulation, where the base joints can have large ranges relative to the arm joints, skewing the sampling distribution away from the manipulation workspace.

I. Failure Cases

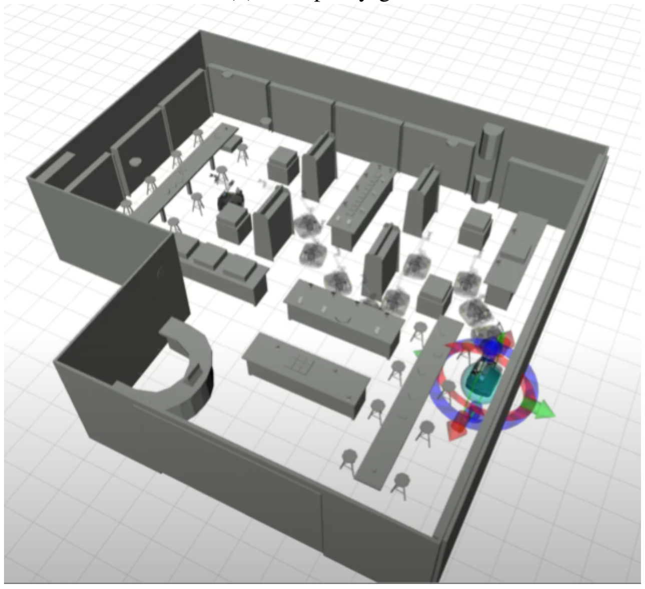
While MGS improves performance in many scenarios, several limitations exist. The root selection procedure computes attractors via BFS in the workspace, which may not reflect configuration-space connectivity; if an obstacle blocks the arm but not the end-effector path, no attractor will be placed to help 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



(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.

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.