Topological Nearest-Neighbor Filtering for Sampling-Based Planners

Read Sandström, Andrew Bregger, Ben Smith, Shawna Thomas, and Nancy M. Amato

Abstract—Nearest-neighbor finding is a major bottleneck for sampling-based motion planning algorithms. The cost of finding nearest neighbors grows with the size of the roadmap, leading to significant slowdowns for problems which require many configurations to find a solution. Prior work has investigated relieving this pressure with quicker computational techniques, such as $kd$-trees or locality-sensitive hashing.

In this work, we investigate an alternative direction for expediting this process based on workspace connectivity. We present an algorithm called Topological Nearest-Neighbor Filtering, which employs a workspace decomposition to select a topologically relevant set of candidate neighbor configurations as a pre-processing step for a nearest-neighbor algorithm. We investigate the application of this filter to several varieties of RRT and demonstrate that the filter improves both nearest-neighbor time and overall planning performance.

I. INTRODUCTION

Sampling-based motion planners such as PRM [1] and RRT [2] explore a problem by sampling random configurations for a robot and connecting them to nearby neighbors. In this context, the nearness of two configurations is determined by a distance metric defined over the entire configuration space. An important component in the process of connecting configurations is a nearest-neighbor search, which determines which existing configuration $q_{old}$ will be tried when connecting to a new sample $q_{new}$.

Nearest-neighbor finding is a major bottleneck for sampling-based planners. A brute-force scan requires linear time for each connection attempt, while more advanced methods such as those based on $kd$-trees [3] approach logarithmic time with a sufficiently large number of configurations. However, prior work has focused primarily on utilizing faster computational techniques as opposed to leveraging the shape of the free space.

In this work, we investigate how workspace connectivity can be employed to identify a reduced set of candidate neighbors for consideration by a nearest-neighbor algorithm. The algorithm, called Topological Nearest-Neighbor Filtering, aims to select a reduced set of candidate neighbors which are topologically relevant to the query configuration. This is realized by leveraging a cell decomposition of the free workspace to define topologically similar neighborhoods of workspace. Roadmap configurations are mapped to these neighborhoods to support quickly locating the configurations in a given neighborhood. When presented with a nearest-neighbor query $q$, the filter locates the set of decomposition cells that are topologically relevant to $q$ and selects only the set of configurations within as candidates for the subsequent nearest-neighbor search.

In this work, we present the filter algorithm and experimental validation demonstrating its usefulness in several environments with four RRT-type planners (RRT [2], SST [4], Dynamic Region-biased RRT [5], and SyCLOP [6]). We show that the filter can improve both nearest-neighbor time and overall planning performance, and also that it can be adapted to bias roadmap connection along any desired flow through a workspace decomposition graph.

II. RELATED WORK

Nearest-neighbor finding has attracted significant attention in sampling-based motion planning research as a primary performance bottleneck. Two primary avenues have been investigated, which are exact and approximate methods. Approximate methods attempt to return a nearby neighbor as opposed to the precise nearest neighbor.

A popular method for exact nearest-neighbor finding in a motion planning context employs a $kd$-tree to quickly compute the nearest neighbor [7]. A later work re-iterates on this method to develop a partitioning strategy which better respects the nuances of the orientation components [8]. The use of $kd$-trees is also observed for nearest-neighbor problems in machine learning. Other structures such as metric and cover trees have also been suggested, although empirical evidence suggests that these are not significantly faster than $kd$-trees in problems of moderate size and dimension [9].

However, other researchers have noted that $kd$-trees perform little better than brute-force when the problem dimension is moderate to large [10]. Such difficulties have spurred investigation in approximate nearest-neighbor methods, which aim to trade a relatively small sacrifice in accuracy for a larger gain in computational speed. Locality-sensitive hashing is a well-known class of approximate methods which attempts to bucket similar samples together with some form of hashing scheme, and has been applied in both machine learning [11], [12] and motion planning [10]. An alternative approximate method from the motion planning realm is distance-based projection onto Euclidean space (DPES), which uses a projection from configuration space to a lower-dimensional Euclidean space before computing the neighbors [13].

The present work relies on a workspace decomposition, which is a partitioning of the free workspace into a set of discrete cells. Decompositions are a well-studied area.
of computational geometry and are covered in standard texts [14]. In this context, we are primarily interested in graph-representations of a workspace decomposition as a means of discovering topological relationships between specific cells which are relevant to the motion planning process.

A prior method which makes use of a workspace decomposition for nonholonomic tree extension is the SyClOPl framework [6]. In this method, the tree is extended by selecting a roadmap configuration \( q \) from the front of a “discrete lead” extracted from a workspace decomposition graph using a search from start to goal. A random control is then applied from \( q \) to propagate the tree forward. This causes the tree to expand preferentially from the outermost samples along the discrete lead; this is effective in practice but cannot be applied to methods which bias the growth direction with sampling.

In this work, we investigate a nearest-neighbor filter based on workspace connectivity that attempts to select the most pertinent configurations for the subsequent nearest-neighbor check. We note that workspace connectivity is an important factor in estimating the probability that a connection will succeed because the local planning algorithms used in sampling-based motion planning are deliberately designed for small steps through configuration space, which implies a small or negligible displacement of the robot body in workspace. Our filter does not require the notion of a query, and can be adapted to respect any desired flow through workspace.

III. Method

We present Topological Nearest-Neighbor Filtering, which is based on three observations:

1) For a given configuration \( q \), other configurations which are not nearby in connected workspace are unlikely to be successfully connected to \( q \) regardless of their \( C_{\text{space}} \) proximity. For example, two configurations on the opposite side of a thin wall are very close in \( C_{\text{space}} \) but quite far apart through connected workspace (Fig. 1).

2) As the roadmap grows in size, the set of configurations that are likely to be connected to \( q \) typically shrinks to a very small fraction of the roadmap.

3) The primary influence on the cost of a nearest-neighbor check is the size of the candidate set.

The filter aims to identify a relatively small set of candidate neighbors for \( q \) which are likely to be successfully connected to \( q \) by examining the workspace topology. We refer to such configurations as “good candidates” for connecting to \( q \).

A convex cell decomposition of the workspace can be leveraged to locate such candidates by maintaining a mapping between decomposition cells and roadmap configurations. Whenever configurations are added to the roadmap, they are also added to this mapping, referred to here as the “decomposition map”.

The overall approach is to use the decomposition map to locate a neighborhood of cells \( C \) that contain good candidate neighbors for \( q \). First, we locate the cell \( c \) that contains \( q \). Then, a limited single-source shortest paths search (Alg. 3) outward from \( c \) through the decomposition graph identifies the neighborhood \( C \) that is near to \( c \) via a path through connected workspace. We say that cells meeting this criteria are “topologically relevant” when connecting to configurations in \( c \). Once the neighborhood has been identified, the decomposition map provides the set of roadmap configurations that reside within. These configurations are passed to the nearest-neighbor algorithm as candidate nearest-neighbors.

The filter’s role in planning is thus to minimize the work that the nearest-neighbor algorithm must perform.

A. Finding the Decomposition Cells

To find the decomposition cell that holds a particular configuration \( q \), we use the spatial degrees of freedom as a reference point \( p \) and say that \( q \) is “contained” by the cell \( c \) which contains \( p \) (as in [6]). The cell \( c \) can be located in time polylogarithmic in the number of decomposition cells [14], although in practice a simpler implementation is often adequate.

A two-stage search can provide effective results: first, impose a uniform grid over the workspace and locate the voxel \( g \) containing \( q \) in constant time. Then, check each decomposition cell which contacts \( g \) and return the one that contains \( q \) (Alg. 1). Both the grid and the mapping from voxel to touched cells are computed once as a pre-processing step, so the second stage takes time linear in the maximum number of decomposition cells that touch a given voxel, which is usually very small for a well-formed decomposition.

Algorithm 1 Find the decomposition cell associated with a designated point. The set of cells touching each grid voxel is computed once as a pre-processing step.

<table>
<thead>
<tr>
<th>Algorithm 1</th>
<th>Find the decomposition cell associated with a designated point. The set of cells touching each grid voxel is computed once as a pre-processing step.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: function FINDWORKSPACECELL(Configuration ( q ))</td>
<td>2: ( g \leftarrow \text{FINDGRIDVOXEL}(q) )</td>
</tr>
<tr>
<td>3: for all ( cell \in g ).GETTOUCHINGCELLS( )</td>
<td>4: if ( q \in cell )</td>
</tr>
<tr>
<td>5: return ( cell )</td>
<td>( q ) must be in some ( cell )</td>
</tr>
</tbody>
</table>

B. Mapping Decomposition Cells to Roadmap Vertices

While constructing the roadmap, a decomposition map is also created to map roadmap vertices to their containing
cells and vice versa. Whenever a new vertex \( q \) is added to the roadmap, the process described above determines the containing cell \( c \). The decomposition map is updated to make this information readily available for neighbor filtering (Alg. 2). After locating the workspace cell, the cell map update can be implemented in amortized constant time using a hash map from cells to vectors of configurations.

### Algorithm 2 Updating and querying the cell map. We imply global structures for the maps to minimize parameters in this presentation, but this is not a requirement.

1. function \( \text{UPDATECELLMAP}(\text{Configuration} \ q) \)
2. \( \quad \text{cell} \leftarrow \text{FINDWORKSPACECELL}(q) \)
3. \( \quad \text{cfgsInCell}[\text{cell}] \leftarrow \text{cfgsInCell}[\text{cell}] \cup \{q\} \)
4. \( \quad \text{cellContainingCFG}[q] \leftarrow \text{cell} \)
5. function \( \text{ISPOPULATED}(\text{Cell} \ c) \)
6. \( \quad \text{return} \ \text{cfgsInCell}[c] \) is not empty

Note that for planning methods which delete configurations from the roadmap, the deleted configurations must also be removed from the decomposition map. This can also be supported in amortized constant time with a hash map implementation for the map from configuration to decomposition cell.

### C. Finding Candidate Neighbor Vertices

The decomposition cell \( c \) containing a sample configuration \( q \) identifies the immediate region of workspace containing \( q \). Assuming we could determine some neighborhood of cells \( C \) that are topologically relevant to \( c \), the cell mapping can be employed to find a set of candidate neighbors for \( q \) in the neighborhood \( C \).

We now require a means to determine the neighborhood of other decomposition cells \( C \) that are topologically relevant to \( c \). This set will be referred to as the candidate cells for \( c \).

### D. Finding Candidate Cells

For any cell \( c \), we can find the set of topologically relevant candidate cells \( C \) by conducting a single-source shortest paths search on the decomposition graph outward from \( c \) and stopping when some threshold distance \( \delta_{\text{threshold}} \) is exceeded (Alg. 3).

The discovered cells are topologically relevant to \( c \) because the search follows the connected free space, and they are nearby because we enforce a maximum expansion distance \( \delta_{\text{threshold}} \).

### Algorithm 3 Single-source shortest paths (SSSP) interface.

Run an SSSP algorithm using a reduced set \( M \) of the decomposition graph’s adjacency map, and terminate when \( t \) is true. Return a map from each discovered cell to its score (distance from source) and successors.

1. function \( \text{SSSP}(\text{Cell} \ c, \ \text{AdjacencyMap} \ M, \ \text{Termination-Criteria} \ t = \text{none}) \)
2. \( \quad \text{return} \ \text{map from cell to score and successors} \)

Given two adjacent cells \( a \) and \( b \), the line between their segments may cross into obstacle space, depending on the type of decomposition used and the shape of the workspace (Fig. 2a). As such, the free space distance between their centers is not necessarily equal to the Euclidean distance between cell centers. A simple approximation for the free-space distance is to measure the Euclidean distance along the line segments from the centers of \( a \), \( b \) to the midpoint of the shared facet between them. Using this edge weighting avoids measuring virtual short-cuts through obstacle space which are not physically realizable.

### E. The Sampling Frontier

While the above method locates good candidate cells for some query cell \( c \), it is entirely possible that there are no roadmap configurations within this neighborhood. In this case, we must look further outward to find a relaxed neighborhood that does contain configurations.

The nearest such neighborhood is the “sampling frontier” \( F \) of grid cells that are populated and reachable from \( c \) by traversing only unpopulated cells. This represents the subset of the roadmap that is topologically nearest to \( c \) because any other configurations would have to connect through one of these regions to reach \( c \). This neighborhood is different from the overall roadmap frontier in that \( c \) may be embedded within obstacles or within a gap in free space coverage. It follows that \( F \) will be at most the entire roadmap frontier and may be significantly smaller.

The neighborhood \( F \) may be very large and far from the original cell \( c \) if \( c \) is far from the covered free space, and would seem to be not topologically relevant to \( c \) in that case. However, there is no set of candidate cells with better topological relevance, and the set of configurations in \( F \) will typically be much smaller than the entire roadmap.

It is also possible that there are populated cells far from \( c \) which are connected by an unpopulated path, and also populated cells much closer to \( c \) (i.e., as in Fig. 2b). In this case, work would be wasted in discovering the entire frontier after the most useful portion has already been identified.
initially, but as the roadmap coverage of free space increases, it is more and more likely that the filter will be terminated upon visiting a cell at least the first populated cell at distance\( \delta = \delta_{\text{threshold}} \) greater than the first populated cell.

To avoid this problem, we can select a limited frontier instead by terminating the search early. After visiting the first populated cell at distance \( D \) from the source, the search will be terminated upon visiting a cell at least \( D + \delta_{\text{threshold}} \) from the source. This selects only the most promising cells as the limited sampling frontier (Alg. 4).

This frontier may be empty if there are no roadmap configurations in any cell that is connected to \( c \) - in this case, there are no connectable configurations in the roadmap. A non-empty frontier is guaranteed to hold at least one configuration by construction, which will be relatively nearby through connected workspace compared to other configurations in the roadmap.

The worst case complexity for locating the limited frontier occurs when there are no configurations in the roadmap. In this case, the search executes a complete single-source shortest paths algorithm over the entire set of cells reachable from \( c \) in the decomposition graph. This may be expensive initially, but as the roadmap coverage of free space increases, it is more and more likely that \( F \) will represent a topologically relevant neighborhood. The limited frontier can also be cached for methods which do not delete vertices from the roadmap, and can be updated in linear time on each reuse. As such, the filter’s efficiency and efficacy improve as the roadmap grows in size.

Nonetheless, it is pertinent to choose a fairly coarse decomposition to minimize the cost of the candidate cell search. Extremely fine decompositions provide more stratification of neighborhoods than is needed for the filter, and offer little benefit in return for the additional overhead. A good heuristic is to choose a convex decomposition that generates the smallest number of well-formed cells needed to cover the environment.

### F. Query Relevance

For tree-based methods, choosing neighbors that are relevant to solving the query is equally important because performance depends on choosing cells that are likely to generate productive extensions from an “earlier” region of the problem (i.e., closer to the query start) towards a “later” region (i.e., closer to the query goal). We refer to candidate cells meeting this criteria as “query relevant”.

The filter can be adjusted to consider the query relevance of potential candidate cells by determining a subset of the original adjacency map for the decomposition graph which discards edges leading away from the goal. This can be computed as a pre-processing step with a single-source shortest paths algorithm starting from the cell which contains the query goal. The resulting distances (which we will call scores) are a measure of each cell’s proximity to the goal through connected workspace: smaller scores indicate closer proximity. Adding the cross-edges creates an adjacency map where each cell’s successors are of equal or greater distance from the query (Alg. 4).

The filter will now use this mapping instead of the original when exploring for the sampling frontier. This further limits the frontier to those cells that are “behind” the original cell \( c \) relative to the goal to encourage productive extensions. The method could feasibly be applied with other types of adjacency maps for specialized problems.

### IV. Theoretical Intuition

In topology, a **cover** of a topological space \( U \) is a collection \( \mathcal{U} \) \( \subseteq U \) such that \( \cup \mathcal{U} = U \). Our decomposition cells represent a cover \( C \) of the free workspace \( W_{\text{free}} \) where each cell \( c_i \in C \) is convex, and for \( c_a, c_b \in C \), \( c_a \cap c_b \neq \emptyset \) iff \( c_a \) is adjacent to \( c_b \) in the decomposition graph \( G \). Thus, the adjacency relationships in \( G \) define the 1-neighborhood of each cell \( c_i \in C \).

In a sense, \( G \) is a coarse chart of \( W_{\text{free}} \). This is useful because \( W_{\text{free}} \) is a reasonable over-estimation of \( W_{\text{free}} = \text{projection of } C_{\text{free}} \text{ into } W \), so \( G \) is also a coarse, approximate chart of \( W_{\text{free}} \). It is approximate in the sense that it contains all of \( W_{\text{free}} \) as well as some portions of \( W_{\text{free}} \) which do not intersect \( C_{\text{free}} \). As such, two configurations \( q_a, q_b \in c \) for \( c \in C \) are within some convex neighborhood in our approximation of \( W_{\text{free}} \). This is a necessary but not sufficient condition for connecting \( q_a, q_b \) via a straight line in \( C_{\text{free}} \); it does not guarantee that \( q_a, q_b \) are connected, but it is a much better estimator than the Euclidean distance \( ||q_a - q_b|| \).

---

**Algorithm 4** Find a sampling frontier.

1: ***General frontier finding.***
2: function FINDSAMPLINGFRONTIER(Cell cell, AdjacencyMap map)
3:   stop ← visited a cell with distance at least \( \delta_{\text{threshold}} \)
4:   childMap ← SSSP(cell, m, stop)
5:   ***Pick out the frontier.***
6:   frontier ← \{ \}
7:   for all parent ∈ childMap.keys
8:       if ISPISOPULATED(parent)
9:          frontier ← frontier ∪ \{parent\}
10:  return frontier
11: ***Topological relevance only.***
12: function FINDSAMPLINGFRONTIER(Cell cell)
13:   map ← Original adjacency map
14:  return FINDSAMPLINGFRONTIER(cell, map)
15: ***Topological and query relevance.***
16: function FINDQUERYSAMPLINGFRONTIER(Cell cell)
17:   ***Initialize a query-relevant adjacency map.***
18:     if map is empty or goal has changed
19:        map, ← Original adjacency map
20:       map ← SSSP(goal, map)
21:     ***Add cross edges.***
22:     for all \( v \in \text{decomposition.vertices} \)
23:       for all \( \text{adj} \in v\text{.neighbors} \)
24:         if map[\text{adj}].score < map[\text{v}].score
25:            map[\text{v}].successors ← \text{adj} \cup map[\text{v}].successors
26:  return FINDSAMPLINGFRONTIER(cell, map)
We leverage this heuristic when searching for candidate cells for some sampled configuration \( q_{\text{rand}} \); because \( C \) is a coarsely approximated chart of \( W'_{\text{free}} \), the configurations within the candidate cells are approximately the subset of the sampling frontier which are nearest to \( q_{\text{rand}} \) through \( W'_{\text{free}} \). The heuristic predicts that these are the most promising configurations for attempting a connection to \( q_{\text{rand}} \).

V. Applicability

As it is based on a workspace decomposition, the filter is appropriate for problems where the translational components of \( C_{\text{space}} \) play a significant role in feasible solutions. This includes many types of rigid-body motions, but is not directly applicable to problems with degrees of freedom in only orientation or joint space.

The filter also relies on the assumption that \( C_{\text{space}} \) is locally connected, which holds for many problems in the physical world. It is not expected to perform well in more abstract configuration spaces where a workspace decomposition does not represent an interesting submanifold of \( C_{\text{space}} \). This might be rectified by replacing the workspace decomposition with some other coarse cover of an interesting submanifold of \( C_{\text{space}} \); this investigation is left as future work.

VI. Demonstration

We evaluate the method by comparing nearest-neighbor and planning times for several RRT methods with and without the filter and query relevance. The planners used include RRT [2], SST [4], Dynamic Region-biased RRT [5] (DRB-RRT), SyCLOP [6], and a modified version of SyCLOP for holonomic problems referred to as SyCLOP-holo. These were selected as a spectrum of different levels of heuristic guidance. The unfiltered methods use a brute-force nearest-neighbor search to isolate the gains produced by the filter.

The modified SyCLOP-holo algorithm is a modification of SyCLOP for holonomic problems, in which there are no controls to sample. This variant is identical to the original method except that (a) a sampled configuration \( q_{\text{rand}} \) is used as the growth target as in standard RRT, and (b) once a workspace cell is selected from the discrete lead, we use a traditional neighborhood finder to select \( q_{\text{near}} \) from the set of configurations within (rather than using the selection history as in the original method). These changes aim to adapt SyCLOP to holonomic problems while retaining the spirit of the method. Since SyCLOP and SyCLOP-holo are already choosing \( q_{\text{near}} \) from a single decomposition cell, it does not make sense to subsequently apply our filter. We have included them for comparison because they also employ a decomposition to locate configurations for extension.

A. Experimental Setup

We use four simulated environments for the evaluation with different aspects of interest (Fig. 3). Helico is relatively open. LTunnel presents three narrow entrances. Garage has a four-story winding ramp and additional longer routes toward the other end of the environment. The GridMaze environment has long, winding paths in a cramped tunnel that constrains the robot’s rotational DOFs. Maze-like problems are notoriously difficult for RRT’s because the sampled target configurations \( q_{\text{rand}} \) are quite frequently located across the maze walls, resulting in short, erratic extensions that scrape very close to the obstacle space. Additionally, corners and tight turns frequently create portions of \( C_{\text{space}} \) where only a very small portion of the local sampling volume can yield a configuration that extends the tree around the corner.

Nonholonomic trials were performed in the Helico and LTunnel environments. Nonholonomic robots increase the problem difficulty by increasing the dimensions of the planning space and severely limiting the allowed actions to the robot’s control set.

The robots in all cases are 6 DOF rigid bodies. The RRT maximum extension distance is set to approximately the bounding sphere radius for the robot, and a tetrahedralization is used for the workspace decomposition. The nonholonomic robots are fully actuated with simple discrete control sets (i.e., over a single extension the robot can exert a force on itself in any one of its position or orientation DOFs). An even mix of best and random controls were used for RRT, SST, and DRB-RRT, while random controls were used for SyCLOP (as it does not use a growth target).

All methods were implemented in a C++ motion planning library developed in the Parasol Lab at Texas A&M University, which uses a distributed graph data structure from the Standard Template Adaptive Parallel Library (STAPL) [15], a C++ library designed for parallel computing. The workspace tetrahedralization was performed with a combination of the TetGen [16] and CGAL [17] libraries.

All experiments were executed on a desktop computer running CentOS 7 with an Intel® Core™ 2 Quad Q9550 CPU at 2.83 GHz, 8 GB of RAM, and the GNU g++ compiler version 4.8.5.

Thirty-five trials are run for each evaluation. Each run is limited to a maximum time of three minutes to complete the queries illustrated in Fig. 3. Executions which do not solve the query in this time are considered failures. We report the success rate for each planner and the average execution and nearest-neighbor times for the successful runs (Fig. 4). Error bars indicate standard deviation in all cases. Statistical significance is measured with Welch’s t-test on the successful trials.

The reported run times do not include the time needed to decompose the workspace; the slowest environment to decompose is the Garage, which takes about one second.

B. Analysis

Topological Filter: A consistent drop in nearest-neighbor time is observed for all planners when using the topological filter without query relevance (in both holonomic and non-holonomic trials). The difference from the unfiltered method is highly significant (\( p_{\text{val}} \leq .01 \)) in all cases except DRB-RRT in GridMaze (\( p_{\text{val}} = .05 \)).

The effect on overall planning time is positive with high confidence (\( p_{\text{val}} \leq .01 \)) in the holonomic trials, excepting
In holonomic LTunnel p while SST degraded with high confidence (pval ≤ .01) for SST in both environments and RRT in LTunnel. The other three cases showed low confidence improvements.

The unguided planners RRT and SST also attained a unilateral increase in success rate, and are able to reliably solve the holonomic GridMaze and Garage problems within the three minute time limit. The guided planner DRB-RRT sees equivalent or better success rate in all cases.

**Query Relevance**: The query relevance option presents mixed results for both nearest-neighbor and planning time. The nearest neighbor time is frequently worse compared with the plain filter. In Helico, small differences are seen for all planners with high confidence (pval ≤ .01) except RRT (pval = .90) and nonholonomic DRB-RRT (pval = .12). In holonomic LTunnel, all methods require more nearest-neighbor time than the plain filter with high confidence (pval ≤ .01) except SST (pval = .77). The nonholonomic LTunnel shows low confidence differences across the board (pval ≥ .23). In GridMaze, all planners showed low confidence differences (pval ≥ .43). In Garage, RRT and SST used significantly less nearest-neighbor time with query relevance (pval ≤ .01), while DRB-RRT showed a high confidence increase (pval ≤ .01).

The overall planning time with the query relevance option is frequently worse than the regular filter, with the most extreme cases occurring for the guided planner DRB-RRT. The nonholonomic Helico trials were mixed: RRT showed a low confidence degradation (pval = .27), SST showed a high confidence improvement (pval ≤ .01), and DRB-RRT showed a high confidence degradation (pval ≤ .01). The nonholonomic trials exhibited the opposite trend: RRT and DRB-RRT improved with low confidence (pval ≥ .69) while SST degraded with high confidence (pval ≤ .01). In holonomic LTunnel we see a unilateral increase in planning time for all planners, with low confidence for SST (pval = .73) and high confidence for RRT and DRB-RRT (pval ≤ .01). In the nonholonomic version RRT attained a slight improvement but with very low confidence (pval = .91), while the other two planners again degraded with low confidence (pval ≥ .29). In GridMaze, the planning time increased over the regular filter with high confidence for RRT (pval ≤ .01) and low confidence for SST and DRB-RRT (pval = .2 and pval = .03, respectively). In Garage, RRT and SST saw large improvements while DRB-RRT performed worse, with high confidence (pval ≤ .01) in all cases.

Success rate with query relevance was roughly equivalent to the regular filter in most cases. Reductions are observed for RRT in GridMaze, DRB-RRT in Garage, and SST in the nonholonomic LTunnel.

The SyClOPl planner shows a small amount of nearest-neighbor time because we attempt to connect configurations which are very near to the goal directly; we found this to be necessary in practice to obtain reasonable results. In the nonholonomic Helico, SyClOPl wastes some effort evenly expanding the tree; however the LTunnel results show that this method is highly effective in less expansive environments.

**C. Discussion**

The nearest-neighbor time for the query relevance variant is generally worse than for the plain topological filter because the more restrictive filter returns no candidates with higher probability, and therefore requires more attempts over the entire execution. For DRB-RRT, this restrictiveness appears to interfere with the guidance heuristic in many cases. In these scenarios, the query relevance is filtering out candidates that were expected by the guidance heuristic. Specifically, DRB-RRT employs moving sampling regions from which new samples are drawn; if the sampling regions are moving perpendicular to the flow prescribed by query relevance, this creates contention where the filter could reject the nearby
Fig. 4. Holonomic experiment results, including average execution and nearest-neighbor times for the trials which solved the query within three minutes, and success rate out of thirty-five trials. Error bars indicate standard deviation. A * indicates that no trials were successful. SyCLoP refers to SyCLoP-holo.
The topological filter demonstrates promise as a method for candidate neighbor reduction. The data shows that it improves the nearest-neighbor time significantly in our test cases, and that the query relevance option can provide additional performance boosts for unguided planners.

Since our method is a filtering pre-step for some other nearest-neighbor algorithm, it may be possible to combine it synergistically with fast computational methods such as kd-trees or locality-sensitive hashing. We leave this investigation as future work.

**REFERENCES**


