Hierarchical Planning With Annotated Skeleton Guidance

Diane Uwacu, *Graduate Student Member, IEEE*, Ananya Yammanuru, *Graduate Student Member, IEEE*, Marco Morales, *Member, IEEE*, and Nancy M. Amato, *Fellow, IEEE*

***Abstract*—We present a hierarchical skeleton-guided motion planning algorithm to guide mobile robots. A good skeleton maps the connectivity of the subspace of c-space containing significant degrees of freedom and is able to guide the planner to find the desired solutions fast. However, sometimes the skeleton does not closely represent the free c-space, which often misleads current skeleton-guided planners. The hierarchical skeleton-guided planning strategy gradually relaxes its reliance on the workspace skeleton as $\mathcal{C}_{space}$ is sampled, thereby incrementally returning a sub-optimal path, a feature that is not guaranteed in the standard skeleton-guided algorithm. Experimental comparisons to the standard skeleton guided planners and other lazy planning strategies show significant improvement in roadmap construction run time while maintaining path quality for multi-query problems in cluttered environments.**

***Index Terms*—Motion planning, path planning, semantic scene understanding.**

I. Introduction

MOTION Planning algorithms are applied to a multitude of applications ranging from robotic navigation [1] to microbiology simulations [2]. The sampling-based probabilistic roadmap method (Basic PRM) [3] introduced in the late nineties efficiently deals with the intractability of motion planning problems by approximating high-dimensional spaces. It led to several innovations to improve its performance [4], [5]. The two main targets for improvement are 1) efficiency in quickly finding a sub-optimal solution, and 2) coverage that leads to reusability for multi-query problems.

We recently introduced Dynamic Region Sampling with RRT [6] and Dynamic Region Sampling with PRM [7], two workspace-guided motion planning strategies, as a reliable way to guide planners, especially in applications where the workspace is closely tied to the planning space. The workspace skeleton graph provides a minimal representation of the connectivity of the free space, on top of which the planning algorithm defines dynamic sampling regions that are guided by the edges of the skeleton. However, both strategies rely heavily on the workspace skeleton by constraining roadmap expansion to the intermediate points of the skeleton edges. These strategies also often waste time exploring regions that the skeleton has already validated.

In this work, we introduce the Hierarchical Annotated Skeleton Planning (HASP) strategy for finding workspace-guided paths in complex environments. The skeleton annotations are additional information about the workspace that make the skeleton a better guide for $\mathcal{C}_{space}$ sampling. Some annotations, like obstacle clearance, can be computed automatically, while others, like expected traffic, can be encoded by an expert. Similar to [7], HASP builds a skeleton-guided roadmap. But, in contrast to the previous work, HASP uses the skeleton to find paths hierarchically. It initially searches for paths that are indicated by the skeleton and easy-to-find, and progressively searches for more difficult paths (those that require more planning) according to problem specifications. Additionally, the HASP method delays validation, similar to [8], to reduce the number of collision detection calls that are required to build a decent roadmap.

Specifically, our contribution is a new skeleton-guided strategy that hierarchically focuses planning effort according to solution acceptance criteria. The algorithm improves its reliance on the workspace skeleton as the environment is explored. Additionally, thanks to skeleton annotations, the strategy can be adapted to fit different applications of motion planning. Our experimental analysis shows how the HASP method achieves improved run time, efficiency, and scalability compared to related sampling-based planning strategies.

II. Preliminaries and Previous Work

The pose of all the components of a robot can be fully determined by its configuration, and the set of all configurations within an environment is the *configuration space*, or $\mathcal{C}_{space}$. The dimensionality of $\mathcal{C}_{space}$ is equivalent to the robot's degrees of freedom (DOFs). $\mathcal{C}_{space}$ is often partitioned into free space ($\mathcal{C}_{free}$) and obstacle space ($\mathcal{C}_{obst}$).

Given a start q_s and a goal q_g configurations or regions, that define a query, the motion planning problem consists of finding

Manuscript received 24 February 2022; accepted 14 July 2022. Date of publication 5 August 2022; date of current version 25 August 2022. This letter was recommended for publication by Associate Editor T.-Chiu Au and Editor H. Kurniawati upon evaluation of the reviewers' comments. This work was supported in part by IBM-Illinois Discovery Accelerator Institute and in part by Henry Luce Foundation. *(Corresponding author: Ananya Yammanuru.)*

Diane Uwacu is with the Department of Computer Science and Engineering, Texas A&M University, College Station, TX 77840 USA (e-mail: duwacu@tamu.edu).

Ananya Yammanuru and Nancy M. Amato are with the Department of Computer Science, University of Illinois, Champaign, IL 61801 USA (e-mail: ananyay2@illinois.edu; namato@illinois.edu).

Marco Morales is with the Department of Computer Science, University of Illinois, Champaign USA, and also with the Department of Computer Science, Instituto Tecnológico Autónomo de México (ITAM), Mexico USA (e-mail: moralesa@illinois.edu).

Digital Object Identifier 10.1109/LRA.2022.3196885

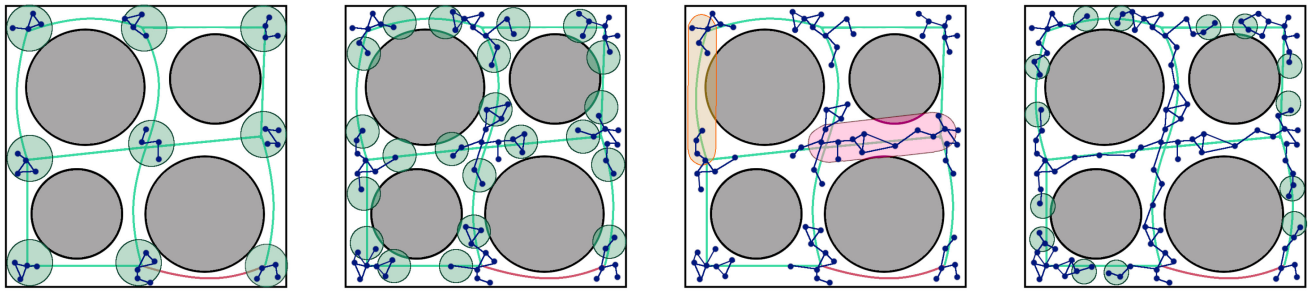


Fig. 1. An illustration of Dynamic Region sampling with PRM. Obstacles are shown in gray. The workspace skeleton is shown in purple. (a) The algorithm samples initial connected components (blue) in regions (green) around each skeleton vertex. (b) Sampling regions expand outward along the skeleton edges. We depict the regions in the location where samples were generated for clarity; in the actual algorithm the regions advance past the newly generated samples. (c) An illustration of two edge segments. The red-shaded segment has a single local component which is also a bridge. The orange-shaded segment has two distinct local components. (d) The components in the middle tunnels successfully connect to form bridges, and their regions are released. The outer passages are still expanding.

a continuous path τ in $\mathcal{C}_{free}$ between the start q_s and the goal q_g as follows:

$$\tau : \mathbb{R} \rightarrow \mathcal{C}_{free} \mid \ \tau(0) = q_s,\ \tau(1) = q_g \tag{1}$$

Where $\tau(0)$ is the initial value and $\tau(1)$ the last.

Generally, a representation of $\mathcal{C}_{obst}$ cannot be computed. The validity of a single configuration q can be determined with the help of collision detection methods. This allows sampling-based motion planning strategies to randomly explore $\mathcal{C}_{space}$ without having to compute $\mathcal{C}_{obst}$.

A. Sampling-Based Algorithms

Sampling based algorithms approximate $\mathcal{C}_{space}$ by generating random robot configurations and connecting them to build a roadmap graph (like the probabilistic roadmap (Basic PRM) [3]) or a tree (like the rapidly-exploring random trees (RRT) [9]). In this work we focus on Basic PRM and its variants whose roadmaps can be searched to find valid paths for multiple queries.

Because it explores an environment randomly, the Basic PRM algorithm suffers from the narrow passage problem in environments with low expansiveness [10]. This problem has been typically mitigated by biasing sampling. For example, obstacle-based algorithms [4], [11], [12] sample close to obstacle surfaces, and medial axis based algorithms [5], [13], [14] sample along the medial axis of the environment and find paths with maximized obstacle clearance. Unfortunately, these strategies are often costly because they require more collision detection calls to push configurations to the desired positions.

B. Guided Motion Planning

To maximize the chance of sampling free $\mathcal{C}_{space}$, guided planning has gained traction. A User-Guided Planning Strategy [15] allows the user to define and manipulate workspace sampling regions that the planner follows in real time to explore the planning space inside the user-defined region. The planner relies on the user's intuition to plan inside narrow passages and find intuitive paths faster. The success of this approach inspired solutions that use topological workspace guidance.

Workspace guidance involves using the workspace structure to direct $\mathcal{C}_{space}$ exploration. Some approaches [16], [17] decompose the workspace to specifically target narrow passages. They partition the workspace into cells that can be used to bias the sampling process. Unfortunately, these methods generally suffer from oversampling because the cells are predetermined.

Workspace skeletons provide better guidance for sampling. A workspace skeleton is a uni-dimensional graph in the workspace such that the free space can be collapsed into the skeleton continuously [18]. The skeleton's vertices are mapped to topological regions of the workspace and the skeleton's edges indicate the connectivity of two adjacent regions. As a result, the skeleton can be computed using computational geometry algorithms. The medial-axis skeleton [19] is a good guide for sampling in 2-dimensional environments, while the mean-curvature skeleton [20] can be used to guide sampling in 3-dimensional environments. Alternatively, an expert could manually produce a skeleton guide specific for a problem.

Our group has used workspace skeletons to guide RRT [6] and PRM [7] algorithms. These strategies guide the planner to sample inside dynamic regions created along the workspace skeleton's edges. The Dynamic Region Sampling with RRT strategy creates a rapidly-exploring random tree along the skeleton, while the Dynamic Region Sampling with PRM method builds a probabilistic roadmap [7] using the skeleton. The experimental results from both methods show that skeleton-guided planners are more efficient than their non-guided counterparts, but their performance depends on the quality of the workspace skeleton. By guiding the sampling regions along the skeleton edges, these algorithms can only find a solution in reasonable time if the solution exists in workspace and is mapped by the skeleton. In addition, these planners are only guided by the workspace connectivity indicated by the skeleton, and do not fully utilize properties specific to the workspace that could be relevant to the exploration of $\mathcal{C}_{space}$. In contrast, in this work we propose to use the information on skeleton edges to limit the level of local exploration done in the connecting regions.

Dynamic Region Sampling with PRM [7], the predecessor of this work (illustrated in Fig. 1) creates local components

Algorithm 1: Hierarchical Annotated Skeleton Planner.

Input: Environment env, Start s, Goal g, Annotated Workspace Skeleton aws

1: $rdmp \leftarrow$ `BuildRoadmap`(aws)
2: **while** $True$ **do**
3: $pathSet \leftarrow$ `BuildPathSet`$(rdmp, s, g, pathSet)$
4: **if** $pathFound$ **then**
5: **return** $pathSet$
6: **end if**
7: **while** $\neg pathSet.empty()$ **do**
8: $nextBestPath \leftarrow pathSet$.`pop`$()$
9: $pathFound \leftarrow$ `FixPath`$(nextBestPath, policy)$
10: **if** $pathFound$ **then**
11: **return** $nextBestPath$
12: **end if**
13: `UpdatePathSet`$(rdmp, s, g, pathSet)$
14: **end while**
15: **end while**

Algorithm 2: BuildPathSet.

Input: Roadmap $rdmp$, Start s, Goal g, Number of desired paths before fixing $maxPaths$, Factor of maximum cost limit ϵ, Where to save paths $pathSet$

1: $paths \leftarrow$ `QueryRoadmap`$(rdmp)$
2: $maxCost \leftarrow (\epsilon \times paths.last.cost)$
3: $p\_cost = 0$
4: **while** $pathSet.size() < maxPaths$ **and** $p\_cost < maxCost$ **do**
5: $nextBestPath \leftarrow paths$.`pop`$()$
6: **if** `isValid`$(nextBestPath)$ **then**
7: $pathSet \leftarrow nextBestPath$
8: **return** $true$
9: **end if**
10: $p\_cost = nextBestPath.cost$
11: $pathSet$.`Add`$(nextBestPath)$
12: **end while**
13: **return** false

around the skeleton vertices and expand them along skeleton edges. Sampling regions are initialized at every skeleton vertex. The $\mathcal{C}_{space}$ inside those regions is explored by generating local components (Fig. 1(a)), and the regions are advanced along the intermediates of the skeleton edges as the local components expand. Regions expand incrementally, making "small steps' from the skeleton vertex towards the center of the edge (Fig. 1(b)). When regions begin to touch or overlap, a bridge connection is attempted, joining the two region (Fig. 1(c)). It is possible that a bridge cannot be made, resulting in two separate paths over the length of the skeleton edge.

III. Method

Algorithm 1 and Fig. 2 describe our proposed HASP method. It receives as inputs the environment, the start and goal configurations, and a topological skeleton of its workspace $aws = (aws_V, aws_E)$ annotated with information that may be relevant to the $\mathcal{C}_{space}$. The user decides on an annotation policy to gather information from the workspace that helps guide the planner in a desired $\mathcal{C}_{space}$ exploration behavior. For example, in some problems, paths that maximize obstacle clearance are preferred [13]. So, as illustrated in Fig. 2(a), each skeleton vertex is annotated with a clearance value while the edges are assigned a weight corresponding to the lowest clearance value along the edge. Algorithm 1 uses the provided skeleton to explore the $\mathcal{C}_{space}$ in three phases described below: building the roadmap, building the path set, and fixing the paths in the path set.

Building the initial roadmap: In Line 1 the roadmap is initialized by sampling configurations at each skeleton vertex that satisfies some solution acceptance criteria. In the case of a clearance-annotated skeleton, the criterion could be having enough clearance for the robot. We then attempt to connect the samples at each of the skeleton vertices. This step is the same as the initialization step from Dynamic Region Sampling with PRM shown in Fig. 1(a). Each of the connected components C corresponding to a skeleton vertex v is called a local connected component, and we retain pairs (C, v) for future reference.

For each edge e of the workspace skeleton with endpoints u and v, if e's weight satisfies acceptance criteria and if local connected components C_u and C_v were generated at u and v, a roadmap edge is drawn to connect the pair (C_u, C_v) without checking for its validity. Such edges are colored in light blue in Fig. 2(a). At the end of this step, we have a partially validated roadmap with local connected components that are fully validated, and connecting them together are roadmap edges that are guided by the validity of the workspace skeleton.

The next steps are done iteratively until a valid path is found or the allocated computing resources are exhausted (Lines 2–15).

Building the Path Set: First, the start and goal configurations are added to the roadmap. In Line 3 of Algorithm 1, the algorithm builds a set of paths from the start to goal configurations.

Algorithm 2 shows how the path set is built. When a potential path is identified, its edges are checked for validity. If a valid path is found, it is returned alone, which triggers the algorithm to exit with success (Lines 4–6 of Algorithm 2). Note that additional policies can be added to the "valid path" check (such as path length or quality requirements) as desired. If on the other hand, the path contains some invalid edges, as shown in Fig. 2(b), it is added to the path set by recording its valid and invalid edges as well as its lower bound cost. The lower bound cost is equivalent to the cost that the path would have if all its edges were assumed valid.

Fixing the path set: Once the path set is built, the planner iterates through the partially-invalid paths in order of least invalid and attempts to fix them on e by one (Lines 7 to 12 of Algorithm 1) using the $FixPath$ method.

In Algorithm 3, the path's invalid edges are fixed in order of importance (Line 1). The default ordering policy is edge length: longer edges have higher priority because they have a

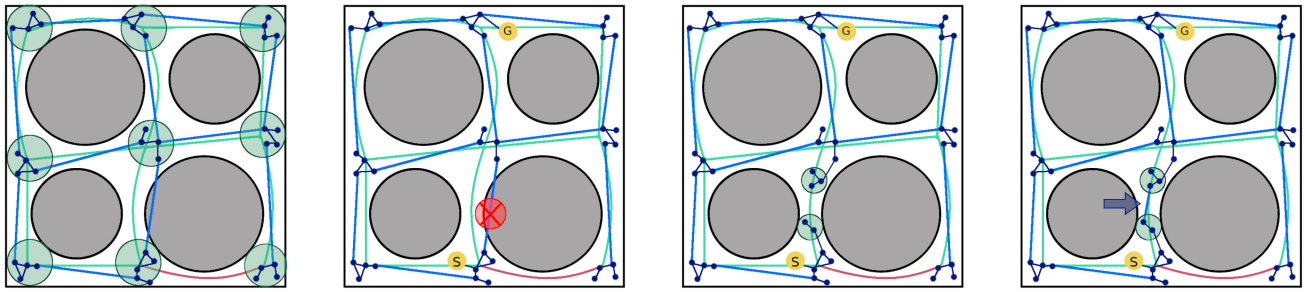


Fig. 2. Example execution of the HASP method: (a) Generate the clearance-annotated workspace skeleton (green edges denote sufficient clearance, red denotes insufficient clearance). Sampling regions are initialized at each skeleton vertex with enough clearance. Edges are added without validation (shown in light blue) if the corresponding skeleton edge weight meets the acceptance criteria (in this example, the skeleton edge is green). (b) Query the roadmap and return all valid paths. If no fully valid paths, mark conflict edges in invalid paths and return. (c) If a valid path was found, return it. Else, attempt to fix invalid edges by expanding sampling regions from each end of the skeleton edge corresponding to the invalid edge. (d) Draw an edge without validation between the expanded sampling regions. Return to step (b).

Algorithm 3: FixPath.

Input: Roadmap $rdmp$, Start s, Goal g, path to fix $path$, policy for fixing path $policy$, where to store unfixable edges $unfixableEdges$

1: $queue \leftarrow$ `OrderEdgesToFix`($policy$) Queue edges in ascending order
2: **for** $edge$ in $queue$ **do**
3: $isEdgeFixed \leftarrow$ `FixEdge`($edge$, $validityCriteria$)
4: **if** !$isEdgeFixed$ **then**
5: $unfixableEdges$.`Add`($edge$)
6: **end if**
7: **end for**
8: **return** $unfixableEdges$.`isEmpty`()

Algorithm 4: UpdatePathSet.

Input: Roadmap $rdmp$, Start s, Goal g, set of paths $pathSet$, list of unfixable edges (edges which weren't able to get fixed in FixPath) $unfixableEdges$.
Output: Updates all paths' invalid edge set.

1: **for** $path$ in $pathSet$ **do**
2: **for** $edge$ in $unfixableEdges$ **do**
3: **if** $edge$ in $path$ **then**
4: $pathSet.remove(path)$
5: **end if**
6: **end for**
7: **end for**

lower chance of being fixed. This makes them good indicators of whether a path is not fixable.

Once the order is determined, the planner attempts to fix each invalid edge (Lines 2–7). The local components from each end of the skeleton edge are expanded towards each other and reconnected with a shorter lazily drawn edge. The steps to fix an edge are illustrated by Figs. 2(c) and 2(d).

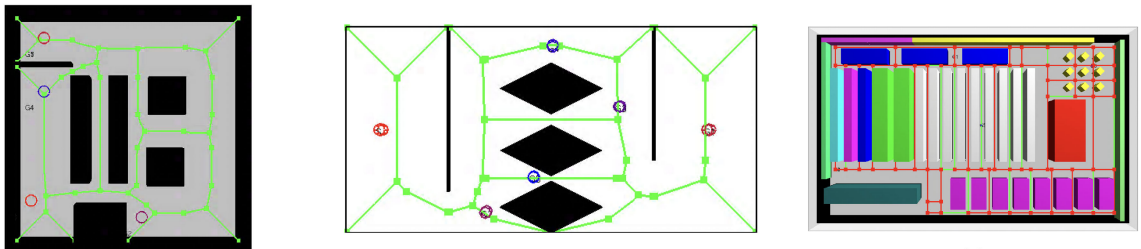
If an invalid edge is not valid after an attempt to fix it, the edge is labeled as "unfixable." The planner updates the path set (Line 13 of Algorithm 1) to remove all paths containing the unfixed edge using Algorithm 4. This prevents the planner from trying to fix an unfixable edge multiple times.

IV. Experimental Validation

We evaluate the performance and solution quality from hierarchically guiding path finding with workspace skeleton guidance. The algorithm is compared to five Probabilistic Roadmaps variants with Basic PRM [3] for baseline. Specifically, we compare HASP to its predecessor, the skeleton-guided Dynamic Region Sampling with PRM [7], and to MAPRM [13] which finds maximum clearance paths. In addition, since HASP performs lazy evaluations, it is compared to the lazy planners Lazy PRM [8], and Partial Lazy PRM, as baselines for postponing validation to the graph search (query) phase. The Partial Lazy PRM method uses partial validation during roadmap construction.

Three problems were selected to validate the algorithm's performance as shown in Fig. 3. The first two were selected to test the ability of the different PRM variants to return the desired path, while the third one tests the scalability of these algorithms.

- *Create* (Fig. 3(a)): A 3 DOF nonholonomic iRobot Create is tested in simulation. In addition to speed, safe navigation is important in this environment with narrow and wide path options. We set the desired path preference to maximum clearance in this environment to study the ability of HASP to return the desired solutions in reasonable time.
- *Rhombus* (Fig. 3(b)): Three rhombuses are placed at different distances from each other to provide pathways with different clearance for a point robot.
- *Store* (Fig. 3(c)): This is a grocery store environment with a 3 DOF nonholomic robot. The aisles in the store environment have different lengths, which highlights the bottleneck of constraining $\mathcal{C}_{space}$ exploration on the intermediates of the workspace skeleton edges. In addition, the environment has fifty obstacles to test the scalability of



(a) (b) (c)

Fig. 3. Testing environments. (a) Create, (b) Rhombus, and (c) Store. The medial axis workspace skeleton is shown in green and the traffic-annotated skeleton is color-coded in the Store environment.

the skeleton-guided algorithms with respect to the collision detection cost.

A. Experiment Setup

In each environment, three queries were defined to study the performance of different PRM variants in multi-query problems. We pre-computed medial-axis skeletons and we annotated them with clearance for all the problems. Their computation took 12 seconds (Rhombus), 16 seconds (Create) and 55 seconds (Store). These times are not reported in the results since they are incurred only once. Each algorithm was given 1000 sampling attempts to build the initial roadmap, and then runs until it solves all queries. All algorithms made two samples per iteration to prevent unnecessarily dense roadmaps and used eight nearest neighbors to connect each sample. We report the time required to build the initial roadmap, the time to solve each query and its path length as a proxy for path cost. In addition, we report the total number of collision detection calls as well as the size of the final roadmap.

The experiments were executed on a desktop computer with an Intel Core i7-3770 CPU at 3.4 GHz, 16 GB of RAM, running CentOS 7 and the GNU g++ compiler version 4.8.5. All methods were implemented in our C++ motion planning library developed in the Parasol Lab at the University of Illinois at Urbana-Champaign. Roadmap validation was done with the PQP-SOLID collision detection method [21]. During roadmap construction, Partial Lazy PRM was set up to use a partially validating collision detection method, RAPID [22]. RAPID only checks for the intersection of object surfaces and doesn't check if an object is completely contained inside the obstacle. Skeletonization for the dynamic region methods was performed with the Medial Axis skeleton [19] implemented in the CGAL library [23]. These models are constructed once and read in with the environment by the planner. The time to build the models was considered pre-processing and not included in the results.

B. Analysis

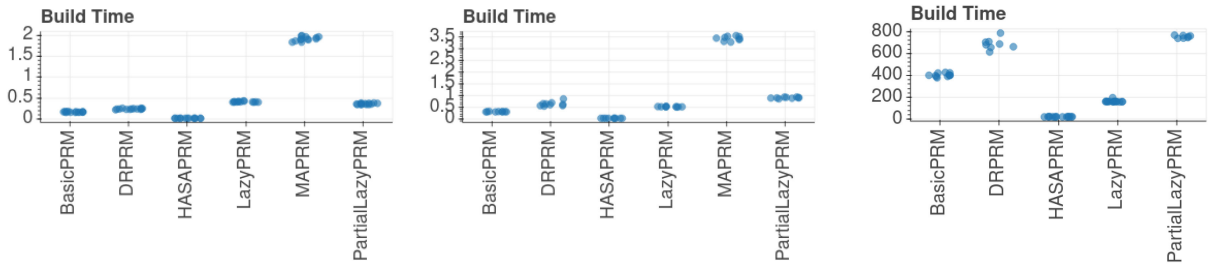
As shown in Table I, HASP has considerably lower roadmap construction time and cost in the three environments. In addition, Table I shows that HASP solves all queries using the smallest roadmap. HASP's number of collision detection calls is second to Lazy PRM's, which reflects the fact that in addition to validating the path, HASP validates its local connected components around skeleton vertices. HASP returns high-clearance paths that are close to the ones returned by MAPRM, thanks to the lesser number of collision detection needed to build a skeleton-guided roadmap.

TABLE I
ROADMAP CONSTRUCTION RESULTS

Environment	Planner	CD calls	Nodes	Edges
Create	Basic-PRM	5,667	599	1,335
	DR-PRM	23,556	255	1,230
	HASP	2,523	**68**	**140**
	Lazy-PRM	**585**	998	15,635
	MAPRM	18,031	539	1,215
	Partial-PRM	49,737	672	9,093
Rhombus	BasicPRM	18,020	817	1798
	DR-PRM	101,678	223	1034
	HASP	11,517	**69**	**138**
	Lazy-PRM	**6,886**	885	12,099
	MAPRM	70,467	892	1949
	Partial-PRM	220,689	817	12,289
Store	Basic-PRM	1,707,796	602	1,505
	DR-PRM	4,706,249	3722	770
	HASP	**1,230,324**	**141**	**360**
	Lazy-PRM	5,396,817	860	10,526
	Partial-PRM	5,096,883	946	6,126

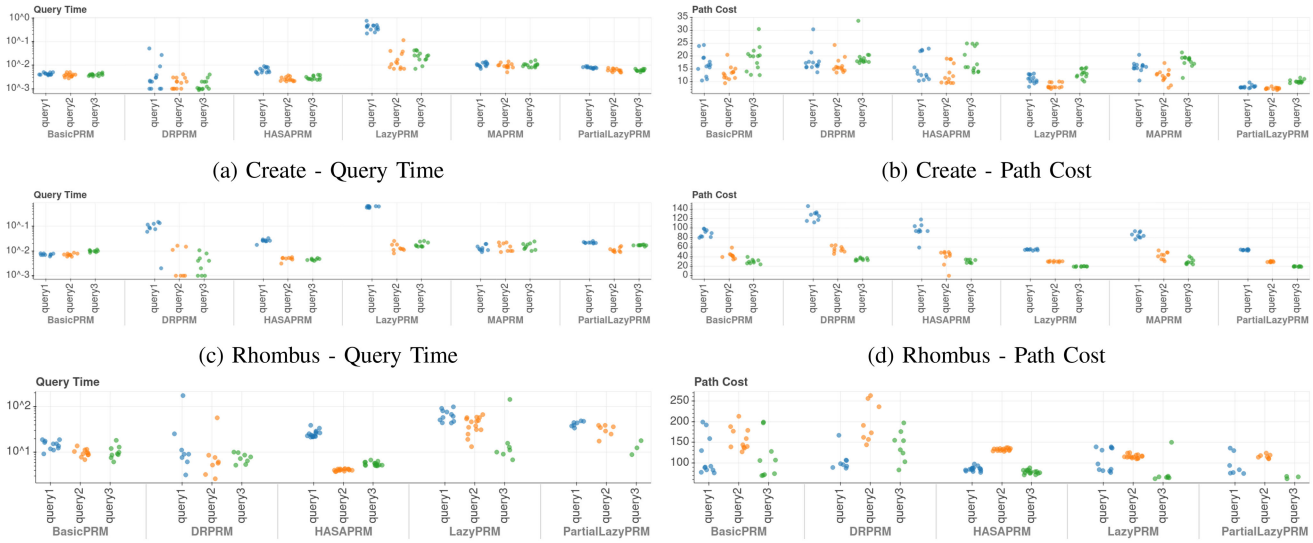
In the Create environment (Fig. 5(a)), HASP returns paths in comparable time to the fully validating algorithms like Basic PRM and Dynamic Region Sampling with PRM. In addition, the path cost is lower than that of most of the paths returned by Dynamic Region Sampling with PRM. In the Rhombus environment (Fig. 5(c)), HASP returns paths in less time than the other lazy algorithms and with lowest path cost. Compared to the other methods, the query results from HASP have the lowest variance. In the Store environment (Fig. 5(e)), MAPRM could not solve any query within the limit of 1000 attempts and 10 minutes of query time.

1) Run Time and Query Time: In all three scenarios in Fig. 4, HASP's planning time is the lowest. Fig. 4(c) shows that as the environment gets more cluttered, HASP's planning time improves by orders of magnitude to that of the other planners. With all the planners in Fig. 5, the query time generally improves as more queries get solved, because the roadmap is progressively improved by the resampling done to solve each query. Fig. 5(a) and 5(c) show that HASP's query time is comparable to that of Basic PRM. In the Rhombus and Store environments, Lazy



(a) Create (b) Rhombus (c) Store

Fig. 4. Run time in seconds.



(a) Create - Query Time (b) Create - Path Cost

(c) Rhombus - Query Time (d) Rhombus - Path Cost

(e) Store - Query Time (f) Store - Path Cost

Fig. 5. Path query time in seconds and path cost.

PRM takes longer to solve the first query, because the original roadmap generated in the planning phase mostly occupied the obstacle space in the narrow passages of the environment. Also, in the Store which is the most complex environment, even if we add the skeleton construction time to the build time for the relevant methods (HASP and Dynamic Region Sampling with PRM), HASP time is still the lowest.

2) Collision Detection Cost: Table I shows that HASP produces sparse roadmaps to solve all the queries. This is correlated with the low number of collision detection calls. In fact, in the Store environment, HASP uses close to five times less collision detection calls than the unguided Lazy PRM. This is because HASP's roadmap does not need as much fixing during the query phase to find a valid path.

3) Scalability: The strength of the hierarchical approach is showcased in the Store environment (Fig. 4(c)) where HASP maintains low planning time as the number of obstacles is increased. In addition, Table I shows that HASP validates its roadmap with less collision detection than Lazy PRM. Given the high number of obstacles in the environment, the planners that skip collision detection usually fare better than their counterparts. However, Lazy PRM suffers in the query time because its roadmap mostly lies in the obstacle space. Moreover, the performance of Dynamic Region Sampling with PRM is degraded in this environment because of its reliance on the skeleton and the high cost of building the roadmap. In fact, Dynamic Region Sampling with PRM did not build a fully connected roadmap in the 1000 attempts allocated to all the strategies, which resulted in more variability in query time and path cost.

4) Dependence on the Skeleton: Planning time and query time in the Store environment show the significance of relying on the skeleton during planning. The long aisles in this environment contain long skeleton edges that constrain Dynamic Region Sampling with PRM to merge its local connected components in more steps by following the intermediates of the skeleton edges. The ability of HASP to merge the local components faster and fix them as needed, proves to be better suited for this kind of environment.

V. Discussion

The results presented in Section IV-B highlight the main strengths of HASP. We note that by skipping the localized exploration of regions mapped by skeleton edges, the strategy

constructs a sparse roadmap of the $\mathcal{C}_{space}$ in record time. Although the resulting roadmap is not fully validated, the query time and path cost results show that skeleton guidance increases the chances of having a roadmap that mostly lies in $\mathcal{C}_{free}$. These results also show that the skeleton, when annotated with properties relevant to the motion planning problem, guides the planner to easily find a desirable path.

Although the presented algorithm was tested in static settings, it could be potentially extended to dynamic settings. The challenges to address for real-time navigation include how to deal with new obstacles in the roadmap structure and in the annotations. Obstacle motions that impact areas not yet validated would not cause any problem, but the validated areas would need to be updated. Also, dynamic environment properties can be marked by the annotations with an efficient updating strategy to avoid stalling. We are currently investigating these motion planning problems.

VI. Conclusion

We present a hierarchical skeleton-guided planning algorithm that initiates local connected components in topological regions of the workspace and connects them using a lazy valid approach and skeleton edge guidance. The method hierarchically queries the roadmap to find easy solutions that are mapped by the workspace skeleton first, fixing the partially valid paths second and searching for the more difficult paths last if needed. Experiments show that HASP is faster than the other PRM variants to construct a roadmap and that it requires the least dense roadmap to find paths with low path cost. In addition, HASP is shown to scale better than the other variants of probabilistic roadmap algorithms, in an environment of fifty obstacles.

This work will be extended to guide multi-robot systems by annotating the skeleton with information relevant to each robot like the traffic patterns of other moving agents in the environment. In addition, this work will be extended to incorporate dynamic environment changes to allow the planner to adapt to changes in the workspace by following the guidance of a dynamically annotated skeleton.

References

[1] B. Englot and F. S. Hover, "Sampling-based coverage path planning for inspection of complex structures," in *Proc. Int. Conf. Automated Plan. and Scheduling*, 2012, pp. 29–37.

[2] D. Brutlag et al., "Using robotics to fold proteins and dock ligands," in *Bioinformatics*, vol. 18, 2002, Art. no. 74.

[3] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Automat.,* vol. 12, no. 4, pp. 566–580, Aug. 1996.

[4] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo, "OBPRM: An obstacle-based PRM for 3D workspaces," in *Proc. Int. Wksp. Alg. Found. Robot.*. Natick, MA, USA: A. K. Peters, Ltd., 1998, pp. 155–168.

[5] C. Holleman and L. E. Kavraki, "A framework for using the workspace medial axis in prm planners," in *Proc. IEEE Int. Conf. Robot. Autom.*, San Franasisco, CA, 2000, vol. 2, pp. 1408–1413.

[6] J. Denny, R. Sandström, A. Bregger, and N. M. Amato, "Dynamic region-biased exploring random trees," in *Algorithmic Foundations of Robotics XII*. Berlin, Germany: Springer, 2020.

[7] R. Sandstrom, D. Uwacu, J. Denny, and N. M. Amato, "Topology-guided roadmap construction with dynamic region sampling," *IEEE Robot. Automat. Lett.*, vol. 5, no. 4, pp. 6161–6168, Oct. 2020.

[8] R. Bohlin and L. E. Kavraki, "Path planning using lazy PRM," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2000, pp. 521–528.

[9] S. M. Lavalle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," *New Directions Algorithmic Computat. Robot.* B. R. Donald, K. Lynch, and D. Rus, Eds. Boston, MA: A. K. Peters, 2001.

[10] D. Hsu, J.-C. Latombe, and H. Kurniawati, "On the probabilistic foundations of probabilistic roadmap planning," *Int. J. Robot. Res.,* vol. 25, pp. 627–643, Jul. 2006.

[11] O. B. Bayazit, G. Song, and N. M. Amato, "Ligand binding with OBPRM and haptic user input: Enhancing automatic motion planning with virtual touch," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, this work was also presented as a poster at RECOMB, 2001, pp. 954–959.

[12] H.-Y. Yeh, S. L. Thomas, D. Eppstein, and N. M. Amato, "UOBPRM: A uniformly distributed obstacle-based PRM," in *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, 2012, pp. 2655–2662. [Online]. Available:http://dx.doi.org/10.1109/IROS.2012.6385875

[13] J.-M. Lien, S. Thomas, and N. Amato, "A general framework for sampling on the medial axis of the free space," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2003, vol. 3, pp. 4439–4444.

[14] Y. Yang and O. Brock, "Adapting the sampling distribution in PRM planners based on an approximated medial axis," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2004, vol. 5, pp. 4405–4410.

[15] J. Denny, R. Sandström, N. Julian, and N. M. Amato, "A region-based strategy for collaborative roadmap construction," in *Alg. Found. Robot. XI*. Berlin, Germany: Springer, 2015, pp. 125–141.

[16] J. Berg and M. Overmars, "Using workspace information as a guide to non-uniform sampling in probabilistic roadmap planners," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2004, pp. 453–460.

[17] H. Kurniawati and D. Hsu, "Workspace importance sampling for probabilistic roadmap planning," in *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, 2004, vol. 2, pp. 1618–1623.

[18] S. Bhattacharya, M. Likhachev, and V. Kumar, "Topological constraints in search-based robot path planning," *Auton. Robots*, vol. 33, 2012, pp. 273–290.

[19] H. Blum, "A Transformation for Extracting New Descriptors of Shape," in Models for the Perception of Speech and Visual Form, W. Wathen-Dunn, Ed., Cambridge, MA, USA: MIT Press, 1967, pp. 362–380.

[20] A. Tagliasacchi, I. Alhashim, M. Olson, and H. Zhang, "Mean curvature skeletons," in *Proc. Symp. Geom. Proc.*, 2012, vol. 31, pp. 1735–1744.

[21] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha, "Fast proximity queries with swept sphere volumes," University of N. Carolina, Chapel Hill, NC, USA, Tech. Rep. TR99-018, 1999.

[22] S. Gottschalk, M. Lin, and D. Manocha, "Obb-tree: A hierarchical structure for rapid interference detection," University of N. Carolina, Chapel Hill, NC, USA, Tech. Rep. TR96-013, 1996.

[23] A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, and S. Schönherr, "On the design of CGAL a computational geometry algorithms library," *Softw. Pract. Exp.,* vol. 30, no. 11, pp. 1167–1202, 2000.