Motion Planning with Dynamics by a Synergistic Combination of Layers of Planning

Erion Plaku Lydia E. Kavraki Moshe Y. Vardi

***Abstract*—To efficiently solve challenging motion-planning problems with dynamics, this paper proposes treating motion planning not just as a search problem in a continuous space but as a search problem in a hybrid space consisting of discrete and continuous components. A multi-layered framework is presented, Synergistic Combination of Layers of Planning (SyCLoP), which combines discrete search and sampling-based motion planning. Discrete search uses a workspace decomposition to compute leads, i.e., sequences of neighboring regions that guide sampling-based motion planning during the state-space exploration. In return, information gathered by motion planning, such as progress made, is fed back to the discrete search. This combination allows SyCLoP to identify new directions to lead the exploration toward the goal, making it possible to efficiently find solutions even when other planners get stuck. Simulation experiments with dynamical models of ground and flying vehicles demonstrate that the combination of discrete search and motion planning in SyCLoP offers significant advantages. In fact, speedups of up to two orders of magnitude were obtained for all the sampling-based motion planners used as the continuous layer of SyCLoP.**

I. Introduction

In motion planning with dynamics the objective is to compute a trajectory to the goal region that not only avoids collisions with obstacles but also satisfies differential constraints imposed by robot dynamics. It is motivated by navigation, exploration, search-and-rescue missions, and other applications, where it is essential to compute trajectories that can be followed by the robot in the physical world.

Motion planning in its early years did not take dynamics into account. Instead, it considered only the geometry of the robot and of the obstacles. Research focused on explicit constructions of the free configuration space, which led to proofs of PSPACE-completeness and to exponential-time algorithms [1]–[5]. This simplified, geometric setting fueled research in sampling-based approaches, giving rise to the Probabilistic RoadMap (PRM) [6], Rapidly-exploring Random Tree (RRT) [7], [8], Expansive Space Tree (EST) [9], [10], and other popular methods (see reviews in [11], [12]).

Sampling-based approaches mark a critical shift from earlier research by departing from explicit constructions of the free configuration space and relying instead on effective sampling of the free configuration space. Sampling-based methods not only were successful in solving high-dimensional geometric problems but also provided a general framework to incorporate dynamics into motion planning by using a tree-based[1] exploration of the state space. The tree is rooted at the initial state and is incrementally extended with trajectories obtained by applying controls and simulating dynamics forward in time. Significant progress has been made, most notably by RRT [7], [8], EST [9], [10], SBL [14], and many other sampling-based motion planners [15]–[24]. However, it has been noted that when dealing with challenging problems with dynamics the tree exploration in many sampling-based motion planners slows down significantly [11], [12], [15], [17], [25], [26].

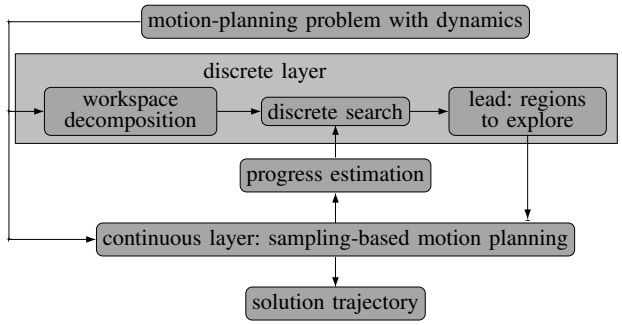


Fig. 1. Proposed multi-layered approach, SyCLoP, synergistically combines discrete search and sampling-based motion planning.

To effectively incorporate dynamics, this paper treats motion planning not just as a search problem in a continuous space but as a search problem in a *hybrid* space consisting of discrete and continuous components. A multi-layered framework is presented, SyCLoP, which combines discrete search and sampling-based motion planning. Fig. 1 provides a schematic representation. The overall effect is that SyCLoP significantly improves the computational efficiency of the underlying sampling-based motion planner. Simulation experiments with dynamical models of ground and flying vehicles showed that when using a sampling-based motion planner as the continuous layer of SyCLoP (see Fig. 1) speedups of up to two orders of magnitude were obtained over the original sampling-based motion planner. These significant speedups were observed for all the sampling-based motion planners (RRT [7], [8], ADDRRT [18], EST [9], [10], SBL [14]) used in the experiments. Advantages offered by SyCLoP become even more pronounced when considering high-dimensional motion-planning problems

Authors are with the Dept. of Computer Science, Rice University, Houston, TX, 77005. E-mail: {plakue,kavraki,vardi}@cs.rice.edu.

Preliminary versions of this work by the same authors appeared in Robotics: Science and Systems, Atlanta, GA, 2007, pp. 326–333 and IEEE Int. Conf. on Robotics and Automation, Pasadena, CA, 2008, pp. 3751–3756. This paper combines the preliminary versions, further improves the proposed framework, and presents results on additional motion-planning problems involving high-dimensional dynamical models of ground and flying vehicles.

Manuscript received February 28, 2011.

[1]A roadmap cannot always incorporate dynamics since each edge (a, b) requires connecting the state a to b via a trajectory that satisfies differential constraints imposed by dynamics. Exact solutions to this steering problem are available only in limited cases, while numerical solutions impose significant computational cost, rendering roadmap construction impractical [13].

with dynamics. Experiments in these cases show that SyCLoP remains efficient, while the computational efficiency of the original sampling-based motion planners deteriorates rapidly as the number of degrees-of-freedom (DOFs) increases.

The objective of the discrete search in SyCLoP is to provide a global sense of direction and identify sequences of regions that the sampling-based motion planner can selectively sample and explore to significantly advance the tree exploration. The motivation for introducing discrete search comes from earlier work in manipulation planning [27], [28] and our recent work on hybrid systems [29]. In manipulation planning, the discrete layer corresponds to a manipulation graph, whose vertices represent stable object placements and whose edges represent transit or transfer actions. In this way, the manipulation graph decomposes manipulation planning into many path-planning problems. Early work [27] used exact motion planners to compute transit and transfer paths, which limited it to manipulators with few joints. To handle high-dimensional manipulators, the work in [28] used discrete search to guide a "fuzzy" PRM in the computation of transit and transfer paths, where fuzzy refers to the probability associated with each roadmap edge as an estimation on the likelihood of the success of the local planner. More recently, similar two-layered approaches have been developed for robot climbing [30], legged motion on varied terrains [31], multi-modal motion planning [32], and rearrangement planning [33], [34]. The other motivation for two-layered approaches comes from hybrid systems, which associate dynamics with each mode and use discrete transitions to switch between modes. Our work [29] showed how to considerably reduce the time to conduct reachability analysis in hybrid systems by using discrete search on the graph of discrete transitions to guide sampling-based motion planning.

Motivated by these results in manipulation planning and hybrid systems, the idea in SyCLoP is to treat motion planning with dynamics as a search problem in a hybrid space composed of continuous and discrete components. While the continuous components correspond naturally to the state space, the discrete components are artificially obtained by imposing a workspace decomposition. Building upon the preliminary implementation of this idea in our earlier DSLX planner [25], [35], SyCLoP significantly improves the proposed framework and efficiently solves high-dimensional motion-planning problems with dynamical models of ground and flying vehicles. The workspace decomposition in SyCLoP is represented as a graph whose vertices are regions of the decomposition and whose edges denote the physical adjacency among the regions. The use of workspace decompositions is motivated by their suitability for problems in mobile robotics, particularly when a challenge for motion planning is in avoiding workspace collisions. Moreover, the low dimensionality of workspace decompositions (two or three dimensions) avoids exponential computational costs associated with state-space decompositions. Decompositions of workspaces, configuration spaces, and state spaces appeared early in motion-planning literature. Key theoretical results and some of the first motion planners were obtained using decompositions [1]–[4]. When sampling-based motion planners became popular, the idea of decompositions has been revisited many times, especially to guide sampling and identify narrow passages in motion planners that construct a roadmap [36]–[41]. More recently, as we review in the related work section, decompositions have been used in tree-based explorations [14], [23], [42]–[45]. Workspace decompositions have also been used in planning for outdoor vehicles [46]–[49]. In this context, a sequence of grid cells or waypoints is first obtained by searching a grid decomposition and then controllers and local collision-avoidance strategies are used to avoid obstacles and follow the waypoints as closely as possible in the physical world.

A crucial aspect of SyCLoP is the close interaction between the discrete search and sampling-based motion planning. The discrete search exploits the simple observation that any solution trajectory corresponds to some sequence of neighboring decomposition regions that starts and ends at regions associated with the initial and goal states, respectively. Such sequence, referred to as a *lead*, provides the sampling-based motion planner with a global sense of direction toward the goal. Sampling-based motion planning can then advance the exploration by extending the tree from one region to its neighbor as indicated by the lead. Note that the decomposition can provide many alternative leads. Due to collision-avoidance requirements and differential constraints imposed by dynamics, in some cases, it may be easy for the sampling-based motion planner to extend the tree from one region to its neighbor in the lead, while in other cases, it may be difficult or even impossible. Then, an issue that arises is which lead to select among the many available alternatives. To address this issue, SyCLoP evaluates the cost of each lead based on information gathered by the sampling-based motion planner, such as progress made in connecting decomposition regions, time spent in exploration, and region coverage. More specifically, for each pair $(\mathcal{R}_i, \mathcal{R}_j)$ of neighboring decomposition regions SyCLoP maintains a running estimate

$$\textsc{cost}(\mathcal{R}_i, \mathcal{R}_j)$$

on the cost of having the sampling-based motion planner spend additional time attempting to extend the tree from $\mathcal{R}_i$ to $\mathcal{R}_j$. In this way, when $\textsc{cost}(\mathcal{R}_i, \mathcal{R}_j)$ is low, SyCLoP estimates that it is worth spending more time exploring $(\mathcal{R}_i, \mathcal{R}_j)$. Drawing from earlier work [25], [35] and extensive experiments, the cost estimates in this paper are designed to be computed efficiently and are shown to work well in practice for solving challenging motion-planning problems with dynamics. Moreover, the same cost estimates are used in this paper for all the different sampling-based motion planning implementations of the continuous layer of SyCLoP (Fig. 1), i.e., RRT, ADDRRT, EST, SBL. The computation of a lead $[\mathcal{R}_{i_j}]_{j=1}^{k}$ then essentially becomes a search algorithm on a weighted graph, where $\textsc{cost}([\mathcal{R}_{i_j}]_{j=1}^{k}) = \textsc{cost}(\mathcal{R}_{i_1}, \mathcal{R}_{i_2}) + \cdots + \textsc{cost}(\mathcal{R}_{i_{k-1}}, \mathcal{R}_{i_k})$. Aiming to strike a balance between greedy and methodical search, SyCLoP selects more frequently low-cost leads, i.e., low $\textsc{cost}([\mathcal{R}_{i_j}]_{j=1}^{k})$, but at the same time it does not ignore other more costly leads. Random leads are also used, although less frequently, as a way to correct for errors inherent with the cost estimates. The computation of leads, exploration by the sampling-based motion planner, and updates of cost estimates make the core loop of SyCLoP:

- Compute the current lead by searching the workspace decomposition. Use the cost estimates to bias the search toward low-cost leads.
- Use sampling-based motion planning to further advance the exploration by extending the tree from one region to its neighbor as indicated by the lead.
- Update cost estimates based on new information gathered by the sampling-based motion planner during exploration.
- Use the updated cost estimates to discover in future iterations new leads that can effectively guide the exploration toward the goal.

The synergistic combination of discrete search and sampling-based motion planning provides SyCLoP with the flexibility to quickly extend the tree toward the goal while able to explore new regions if information from motion planning suggests other less costly leads. This flexibility tends to prevent the sampling-based motion planner in SyCLoP from getting stuck, which is a common problem for many other sampling-based motion planners. As a result of effectively guiding the tree exploration, SyCLoP is shown to efficiently solve challenging motion-planning problems with dynamics.

The paper is as follows. Related work on tree-based exploration in sampling-based motion planning with dynamics is reviewed in Section II. The motion-planning problem with dynamics is defined in Section III. SyCLoP is described in Section IV. Experiments are described in Section V, which also includes a study on the impact of the workspace decomposition. The paper concludes with a discussion in Section VI.

II. Related Work

In motion planning with dynamics, sampling-based methods that use a tree-based exploration of the state space have become the norm [11], [12]. This discussion of related work is divided into two main parts:

- Strategies proposed to guide the tree-based exploration (Section II-A). This discussion is further divided into non-hierarchical strategies (i.e., vertex selection, Section II-A1) and hierarchical strategies (i.e., region selection followed by vertex selection, Section II-A2).
- Strategies proposed to extend the tree by means of trajectory generation (Section II-B). Discussion includes selection of random controls, best controls, motion primitives, and trajectory diversity.

A. Guiding the tree exploration

1) Vertex selection: Each vertex is a possible candidate from which to expand the tree [7]–[10], [16], [18], [21], [50], [51]. RRT [7], [8] first samples a state s uniformly at random and then expands the tree from the vertex that is closest to s according to a distance metric. ADDRRT [18] further improves RRT by dynamically adjusting the sampling domain based on the success and failures of previous expansions. While ADDRRT uses spherical domains, RGRRT [16] uses reachable sets, which are more likely to promote tree expansion under differential constraints. In problems with high-dimensional manipulators, the work in [52], [53] grows an RRT in the task space, which is generally of much lower dimensionality. EST [9], [10] takes a different approach from RRT methods by selecting each vertex v with probability inversely proportional to the density of a small neighborhood around v. Note that RRT and EST approaches rely on nearest neighbors, which can become a bottleneck in high-dimensional problems [54], [55]. Efficient nearest-neighbor implementations for motion planning can be found in OOPSMP [56], [57] and MPNN [55], [58].

2) Region selection: Other sampling-based motion planners guide the tree exploration by first selecting a region and then selecting a vertex [14], [25], [42]–[45]. Regions are generally obtained by decompositions of the workspace, configuration space, or state space. Each time a new vertex is added to the tree, it is also added to the appropriate region. The tree is then expanded by first selecting a non-empty region $\mathcal{R}$ and then selecting a vertex from the tree vertices associated with $\mathcal{R}$. The rationale for this two-level selection scheme is that it reduces the computational time required to make a selection, since the number of non-empty regions is much smaller than the number of tree vertices. Moreover, it allows selections at a region level, which could potentially be useful in determining new directions along which to expand the tree exploration. SBL [14] uses a low-dimensional grid for the decomposition and uniform probability distributions for both region and vertex selection, effectively pushing the tree toward sparse regions. PDST [42] uses a subdivision decomposition based on a low-dimensional projection and selects regions based on a deterministic priority scheme. Extensions of PDST in [44] use a potential field to select the region with the lowest potential. KPIECE [45] constructs a multi-level grid decomposition over user-defined or random linear state-space projections and then selects a region by selecting cells from the top to the bottom grid level.

SyCLoP has several important distinctions from related work. SyCLoP keeps information not only about decomposition regions but also about decomposition edges. This information, which is updated after each exploration step, measures the overall progress and is used to effectively guide the tree-based exploration. The discrete layer (Fig. 1) through the use of leads (sequences of neighboring decomposition regions from initial to goal) provides the sampling-based motion planner with a global sense of direction toward the goal. This plays a significant role in the computational efficiency of SyCLoP since, by exploring regions associated with a lead, the sampling-based motion planner can make rapid progress toward the goal. Moreover, the cost estimates associated with decomposition edges allow SyCLoP to seek alternative leads when it becomes too difficult or impossible to extend the tree from one decomposition region to another. As a result, as shown by the experiments, SyCLoP is capable of avoiding spending valuable computational time exploring and connecting regions that do not advance the search, while at the same time, identifying new directions that can further lead the tree exploration toward the goal.

B. Expanding the tree exploration

After selecting a vertex v for expansion, the exploration tree is extended by generating a trajectory that starts at the

state s associated with the vertex v. A common strategy is to apply some control u to s and simulate the dynamics forward in time until a collision occurs, a state-constraint is violated, or a maximum number of steps is exceeded [11], [12]. Intermediate states along the trajectory are added to the tree, as also suggested in [7], [8], [11], [12], [18], [50]. The control u is generally selected uniformly at random to allow subsequent calls to extend the tree along new directions [10], [14], [25], [42], [45]. To pull the tree toward a randomly sampled state s_{rand}, RRT approaches sometimes try several controls and select the one that brings the state s closer to s_{rand} [8], [18]. To promote expansion in maximally different directions, the work in [50] selects a control that is as different as possible from all the previous controls applied to s. To promote effective exploration toward narrow passages, the work in [20] uses principal-component analysis to compute directions along which to expand the tree. It is also possible to select controls based on motion primitives designed for the specific dynamics under consideration. The work in [59] selects controls based on a maneuver automata, which is specifically designed to take advantage of helicopter dynamics. The work in [42] uses PRM to construct the maneuver automata for certain systems with symmetries. The use of motion primitives typically improves the quality of the solution trajectories. The work in [60], [61] looks not at just one tree trajectory but at all tree trajectories in order to prune the tree into a smaller tree that maintains the overall reachability. This has benefits as a post-processing step to ensure diversity among tree trajectories.

Note that SyCLoP can use many of the above strategies, since different sampling-based motion planners can be used to implement the sampling-based layer of SyCLoP.

III. Problem Statement

A motion-planning problem with dynamics is a tuple $\mathcal{P} = (\mathcal{S}, \mathcal{U}, f, \text{VALID}, s_{init}, s_{goal})$, where

- $\mathcal{S}$ is a state space consisting of a finite set of variables that describe the state of the system;
- $\mathcal{U}$ is a control space consisting of a finite set of input variables that can be applied to the system;
- $f : \mathcal{S} \times \mathcal{U} \to \dot{\mathcal{S}}$ defines the equations of motion as a set of differential equations that describes the differential constraints imposed by the dynamics;
- $\text{VALID} : \mathcal{S} \to \{\texttt{true}, \texttt{false}\}$ specifies constraints which states should satisfy, e.g., collision avoidance, bounds on steering angle, velocity, turning radius.
- $s_{init} \in \mathcal{S}$ is an initial state;
- $s_{goal} \in \mathcal{S}$ is the motion-planning goal. As it is common in motion planning with dynamics literature [11], [12], a state $s \in \mathcal{S}$ is said to satisfy the goal when s is close to s_{goal}, i.e., $\rho(s, s_{goal}) \leq \delta$, where $\rho : \mathcal{S} \times \mathcal{S} \to \mathbb{R}^{\geq 0}$ is a distance metric and $\delta > 0$ is a distance threshold.

The objective is then to compute a control function $\tilde{u} : [0, T] \to \mathcal{U}$ such that the trajectory $\gamma : [0, T] \to \mathcal{S}$ resulting from applying $\tilde{u}$ to s_{init} satisfies the state constraints $(\text{VALID}(\gamma(t)) = \texttt{true}, \forall t \in [0, T])$ and $\gamma(T)$ satisfies the goal. The trajectory γ is obtained by integrating the differential equations $\dot{\gamma}(h) = f(\gamma(h), \tilde{u}(h))$ starting at s_{init}, i.e.,

$$\gamma(t) = s_{init} + \int_0^t f(\gamma(h), \tilde{u}(h))\, dh.$$

IV. SyCLoP

To effectively guide the tree-based exploration of the state space, SyCLoP combines discrete search and sampling-based motion planning. Pseudocode is given in Algo. 1. Sections describing the main steps of the algorithm are referenced at the end of each line.

Algorithm 1 SyCLoP

Input: $\mathcal{P} = (\mathcal{S}, \mathcal{U}, \text{FLOW}, \text{VALID}, s_{init}, s_{goal})$, $\mathcal{W}$: workspace
$t_{\max} \in \mathbb{R}^{>0}$: upper bound on computation time
Output: A solution trajectory or `null`

```
1: D ← WorkspaceDecomposition(W)                                   ◇IV-A1
2: T ← InitializeTree(s_init)
3: while ElapsedTime < t_max do
   ◇discrete layer: guide exploration
4:   [R_ij]_{j=1}^k ← DiscreteLead(D)                              ◇IV-A2
5:   R_avail ← AvailableRegions([R_ij]_{j=1}^k)                   ◇IV-A3
6:   for several times do
7:     R_ℓ ← SelectRegion(R_avail)                                 ◇IV-A4
   ◇continuous sampling-based layer: explore region
8:     for several times do
9:       v ← SelectVertex(R_ℓ.vertices)                            ◇IV-B
10:      ExtendTree(T, R_ℓ, v)                                     ◇IV-B
11:      for each v_new added by ExtendTree do
12:        if v_new.s satisfies the goal then
13:          return Traj(T, v_new)
14:        R_m ← LocateRegion(Proj(v_new.s))
15:        if R_avail.Exists(R_m) = false then
16:          R_avail.Add(R_m)
   ◇update estimates based on new information
17:        UpdateCoverage(R_m, v_new.s)                            ◇IV-C1
18:        UpdateConnections(v_new.parent, v_new)                  ◇IV-C3
19:    if no improvement in coverage or region connections then
20:      if URand(0, 1) ≤ p then
21:        abandon current lead early
22:        goto line 4 to compute new lead
23: return null
```

SyCLoP starts by computing the workspace decomposition $\mathcal{D}$ (Algo. 1:1) and then initializing the exploration tree $\mathcal{T}$ with s_{init} as its root (Algo. 1:2). As the search progresses, $\mathcal{T}$ is extended by adding new vertices. Each vertex $v \in \mathcal{T}$ is associated with a state $s \in \mathcal{S}$, written as $v.s$. Moreover, each vertex v has a backpointer, $v.parent$, to its parent, which indicates that a valid trajectory connects $v.parent.s$ to $v.s$.

The core loop (Algo. 1:3–22) starts by searching the workspace decomposition $\mathcal{D}$ to compute the current lead $[\mathcal{R}_{i_j}]_{j=1}^k$ (Algo. 1:4). Regions in $\left[\mathcal{R}_{i_j}\right]_{j=1}^k$ which have not yet been reached by $\mathcal{T}$ do not contain any tree vertices from which to extend $\mathcal{T}$. For this reason, $\mathcal{R}_{avail}$ maintains a set of nonempty regions, which can be considered for further exploration (Algo. 1:5). SyCLoP then dedicates some time to the exploration of regions in $\mathcal{R}_{avail}$. Proceeding in an iterative fashion (Algo. 1:6), a region $\mathcal{R}_\ell$ is first selected from $\mathcal{R}_{avail}$ (Algo. 1:7). Sampling-based motion planning is then used to further explore $\mathcal{R}_\ell$ by extending several trajectories from the

tree vertices associated with $\mathcal{R}_\ell$ (Algo. 1:8–10). If during exploration of $\mathcal{R}_\ell$ a solution trajectory is found, then SyCLoP terminates successfully (Algo. 1:12–13). Any new decomposition region reached during the exploration of $\mathcal{R}_\ell$ becomes available for selection during the next iteration (Algo. 1:14–16). The new vertices added to $\mathcal{T}$ during exploration of $\mathcal{R}_\ell$ are used to update the cost estimates, such as the region coverage by vertices in $\mathcal{T}$ and connections among regions by edges in $\mathcal{T}$ (Algo. 1:17–18). As a result of updating the cost estimates and reaching new decomposition regions, different regions could be selected from $\mathcal{R}_{avail}$ to be further explored during the remaining iterations dedicated to the current lead.

After exploring the current lead for some time, SyCLoP goes back at the beginning of the core loop to compute a new lead based on the updated cost estimates. SyCLoP may also abandon the current lead early when sampling-based motion planning does not make much progress during the exploration of regions in $\mathcal{R}_{avail}$. In fact, each time the exploration of a region $\mathcal{R}_\ell$ does not improve the overall coverage or region connections, then, with probability p (set to 0.25 in all the experiments), SyCLoP abandons the current lead in favor of a new lead (Algo. 1:19–22). In this way, instead of wasting valuable computational time exploring regions that do not advance the search, SyCLoP seeks alternative leads that could expand the tree exploration along new directions. This flexible interaction between discrete search and sampling-based motion planning, as shown in the experiments, allows SyCLoP to efficiently find solutions to challenging motion-planning problems with dynamics. Details of the main steps in Algo. 1 follow.

A. Discrete Layer

The discrete layer in SyCLoP includes the workspace decomposition (Algo. 1:1), lead computations (Algo. 1:4), and selection of regions that should be further explored by the sampling-based motion planner (Algo. 1:5–7).

1) Workspace Decomposition: Let $\mathcal{W}$ denote the two- or three-dimensional workspace (including the obstacles) on which the robot operates. $\mathcal{W}$ is decomposed into nonoverlapping regions (except at the boundary), i.e., $\mathcal{W} = \mathcal{R}_1 \cup \mathcal{R}_2 \cup \cdots \cup \mathcal{R}_n$, and $\forall \mathcal{R}_i, \mathcal{R}_j \in \mathcal{W} : \textit{Interior}(\mathcal{R}_i) \cap \textit{Interior}(\mathcal{R}_j) = \emptyset$. The physical adjacency of the regions in the decomposition is expressed by a set of edges E, where $(\mathcal{R}_i, \mathcal{R}_j) \in E$ iff $\mathcal{R}_i$ and $\mathcal{R}_j$ are physically adjacent to each other.

To map states to workspace decomposition regions, SyCLoP first uses a projection $\textsc{Proj} : \mathcal{S} \to \mathcal{W}$ to map a state $s \in \mathcal{S}$ to a point in $\mathcal{W}$, i.e., position of the reference point (usually, the centroid) in $\mathcal{W}$ when the robot is at state s. SyCLoP then uses a function $\textsc{LocateRegion} : \mathcal{W} \to \{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$ to map each workspace point to the corresponding region, i.e.,

$$\forall p \in \mathcal{W} : \textsc{LocateRegion}(p) = \mathcal{R}_i \text{ iff } p \in \mathcal{R}_i.$$

In this way, a state $s \in \mathcal{S}$ is mapped to $\mathcal{R}_i$, where $\mathcal{R}_i = \textsc{LocateRegion}(\textsc{Proj}(s))$.

The computation of workspace decompositions is an active research area in computational geometry [62]. Simple decompositions can be obtained by imposing a uniform grid over $\mathcal{W}$, where each cell constitutes a decomposition region $\mathcal{R}_i$. In this case, LocateRegion can be implemented to run in constant time. Other workspace decompositions can be obtained by triangulations. In the case of triangulations, LocateRegion can be implemented to run in polylogarithmic time [62]. The impact of workspace decompositions on the overall computational efficiency of SyCLoP is studied in Section V-E.

The discrete layer also keeps information about the regions associated with the initial and goal states of the motion-planning problem, i.e., $\mathcal{R}_{init} = \textsc{LocateRegion}(\textsc{Proj}(s_{init}))$ and $\mathcal{R}_{goal} = \textsc{LocateRegion}(\textsc{Proj}(s_{goal}))$. Putting it all together, the decomposition is a tuple

$$\mathcal{D} = (\mathcal{W}, \{\mathcal{R}_1, \ldots, \mathcal{R}_n\}, E, \textsc{LocateRegion}, \mathcal{R}_{init}, \mathcal{R}_{goal}).$$

2) Computation of Leads: $\textsc{DiscreteLead}(\mathcal{D})$ (Algo. 1:4) computes the current lead at each iteration of the core loop by searching the workspace decomposition $\mathcal{D}$ for a sequence of neighboring regions $[\mathcal{R}_{i_j}]_{j=1}^k$ connecting $\mathcal{R}_{init}$ to $\mathcal{R}_{goal}$.

With high probability p (set to 0.95 in all the experiments), $\textsc{DiscreteLead}(\mathcal{D})$ computes the current lead by using Dijkstra's shortest-path algorithm, where the edge weights are set to $\textsc{cost}(\mathcal{R}_i, \mathcal{R}_j)$. This allows SyCLoP to bias the selection toward low-cost leads, which indicate that it is worth spending additional time exploring regions in $[\mathcal{R}_{i_j}]_{j=1}^k$, since $[\mathcal{R}_{i_j}]_{j=1}^k$ may effectively guide the exploration toward the goal. As noted in the introduction, $\textsc{cost}(\mathcal{R}_i, \mathcal{R}_j)$ estimates are based on information gathered by the sampling-based motion planner during each exploration of $\mathcal{R}_i$ and $\mathcal{R}_j$. Specifically, $\textsc{cost}(\mathcal{R}_i, \mathcal{R}_j)$ is defined as

$$\textsc{cost}(\mathcal{R}_i, \mathcal{R}_j) = \frac{1 + \textsc{sel}^2(\mathcal{R}_i, \mathcal{R}_j)}{1 + \textsc{conn}^2(\mathcal{R}_i, \mathcal{R}_j)} \, \alpha(\mathcal{R}_i) \, \alpha(\mathcal{R}_j),$$

where, for $k \in \{i, j\}$,

$$\alpha(\mathcal{R}_k) = \frac{1}{(1 + \textsc{cov}(\mathcal{R}_k)) \, \textsc{FreeVol}^4(\mathcal{R}_k)}, \text{ and}$$

- $\textsc{cov}(\mathcal{R}_k)$ estimates the progress made by sampling-based motion planner in covering $\mathcal{R}_k$;
- $\textsc{FreeVol}(\mathcal{R}_k)$ estimates the free volume of $\mathcal{R}_k$;
- $\textsc{conn}(\mathcal{R}_i, \mathcal{R}_j)$ estimates the progress made by sampling-based motion planner in extending $\mathcal{T}$ from $\mathcal{R}_i$ to $\mathcal{R}_j$;
- $\textsc{sel}(\mathcal{R}_i, \mathcal{R}_j)$ counts the number of times $\mathcal{R}_i$ and $\mathcal{R}_j$ have been part of a high-level plan or selected for exploration.

Details related to the definition, efficient implementation and updating of these estimates are provided in Section IV-C.

In this way, $\textsc{cost}(\mathcal{R}_i, \mathcal{R}_j)$ is low when $\mathcal{R}_i$ and $\mathcal{R}_j$ have large free volumes, since in such regions it is more likely for the sampling-based motion planner to extend $\mathcal{T}$. $\textsc{cost}(\mathcal{R}_i, \mathcal{R}_j)$ is also low when $\mathcal{T}$ grows rapidly (which is indicated by low $\textsc{sel}(\mathcal{R}_i, \mathcal{R}_j)$ and high $\textsc{conn}(\mathcal{R}_i, \mathcal{R}_j)$, $\textsc{cov}(\mathcal{R}_i)$, $\textsc{cov}(\mathcal{R}_j)$), since in such cases it is estimated that the sampling-based motion planner will continue to make progress. Note that we have conducted extensive experiments in this paper and in the preliminary work [25] with numerous $\textsc{cost}(\mathcal{R}_i, \mathcal{R}_j)$ estimates. These experiments have shown that SyCLoP works well for a wide variety of $\textsc{cost}(\mathcal{R}_i, \mathcal{R}_j)$ estimates. The particular definition in this paper was chosen as it consistently worked well across all the experiments. In fact, exactly the same estimates have been used for all the experiments in this paper,

which include experiments with several ground and flying vehicles, numerous workspace decompositions, and different sampling-based motion planning implementations (e.g., RRT, ADDRRT, EST, SBL) of the continuous layer (Fig. 1) in SyCLoP.

In addition to shortest-path leads, DISCRETELEAD($\mathcal{D}$) uses random leads, although less frequently, as a way to correct for errors inherent with COST($\mathcal{R}_i, \mathcal{R}_j$) estimates. This is motivated by observations made in [63], where random restarts have been suggested as effective ways to unblock the exploration when sampling-based motion planners get stuck. With small probability $1 - p$, the current lead is computed as a random sequence of neighboring decomposition regions connecting $\mathcal{R}_{init}$ to $\mathcal{R}_{goal}$. The computation is carried out by using depth-first search, where the neighbors are visited in a random order.

3) Computation of Available Regions from the Current Lead: AVAILABLEREGIONS($[\mathcal{R}_{i_j}]_{j=1}^{k}$) (Algo. 1:5) selects from the current lead several nonempty regions as candidates for further exploration. Note that empty regions are not considered since they do not contain any vertices from which to extend $\mathcal{T}$. Initially, $\mathcal{R}_{avail} = \emptyset$. Then, $[\mathcal{R}_{i_j}]_{j=1}^{k}$ is scanned backwards. If $\mathcal{R}_{i_j}$ is nonempty, then $\mathcal{R}_{i_j}$ is added to $\mathcal{R}_{avail}$. When $\mathcal{R}_{i_j}$ is added to $\mathcal{R}_{avail}$, it is also decided (with probability p, set to 0.95 in this work), if other regions should be added to $\mathcal{R}_{avail}$. In this way, only nonempty regions are added to $\mathcal{R}_{avail}$ and preference is given to those regions that appear toward the end of $[\mathcal{R}_{i_j}]_{j=1}^{k}$, since such regions are closer to the goal. As the sampling-based motion planner explores regions in $\mathcal{R}_{avail}$, the tree $\mathcal{T}$ may reach regions that were previously empty. These new regions are then added to $\mathcal{R}_{avail}$ and are made available for selection during the remaining iterations (Algo. 1:14–16).

4) Region Selection: As discussed in related work, several sampling-based motion planners, e.g., [14], [25], [42], [44], [45], rely on region selections to guide the tree exploration. Drawing from this body of research, SELECTREGION($\mathcal{R}_{avail}$) (Algo. 1:7) selects a region $\mathcal{R}_\ell$ from $\mathcal{R}_{avail}$ with probability

$$\frac{\mathcal{R}_\ell.w}{\sum_{\mathcal{R}_j \in \mathcal{R}_{avail}} \mathcal{R}_j.w}, \text{ where}$$

$$\mathcal{R}_\ell.w = \frac{\text{FREEVOL}^4(\mathcal{R}_\ell)}{(1 + \text{COV}(\mathcal{R}_\ell))(1 + (\mathcal{R}_\ell.nsel)^2)}$$

(for more details on the estimates see Section IV-C; $\mathcal{R}_i.nsel$ denotes the number of times $\mathcal{R}_i$ has been selected for exploration in Algo. 1:7). This probabilistic selection scheme gives priority to regions that have high free volume, low coverage, and where the sampling-based motion planner has spent little time in the past. The rationale is, as evidenced by experimental results, that by spending additional time exploring such regions, the sampling-based motion planner could make more progress and increase their coverage. Note that the same weight definition is used for all the experiments in this paper.

B. Continuous Layer: Sampling-based Motion Planning

The sampling-based motion planner explores a region $\mathcal{R}_\ell$ by extending several trajectories from the tree vertices associated with $\mathcal{R}_\ell$ (Algo. 1:8–10). As discussed in related work, different strategies have been proposed for selecting vertices based on nearest neighbors, density distributions, entropy, and many other factors [7]–[10], [14], [16], [18], [21], [50], [51]. This paper contains experiments using many of these strategies.

Once a vertex v has been selected, $\mathcal{T}$ is then extended from v by applying a control and simulating the dynamics forward in time. As it is common in motion planning [8], [10]–[12], numerical integration is used for the simulation. To ensure accuracy, as advocated in the literature, this paper uses Runge-Kutta methods and a small integration step.

The related work section discussed several strategies that can be used to select controls, varying from random controls to motion primitives designed to take advantage of the specific dynamics under consideration. As a result of the exploration, new vertices are added to $\mathcal{T}$. If a new vertex v_{new} satisfies the goal, then a solution trajectory is computed by concatenating the tree trajectories from the root vertex to v_{new} (Algo. 1:12–13). If a new region, $\mathcal{R}_{new}$, is reached, it is then added to $\mathcal{R}_{avail}$, so that it becomes available to be selected for further exploration (Algo. 1:14–16). Moreover, the estimates are also updated to reflect the new information gathered by the sampling-based motion planner, as described next.

C. Definition, Computation, and Updates of Estimates used by the Discrete Layer based on Information gathered by the Continuous Layer

As discussed in Section IV-A2, COST($\mathcal{R}_i, \mathcal{R}_j$) estimates are based on information gathered by the sampling-based motion planner during exploration and are defined in terms of coverage (COV($\mathcal{R}_i$)), free volume (FREEVOL($\mathcal{R}_i$)), connections among regions (CONN($\mathcal{R}_i, \mathcal{R}_j$)), and number of previous selections (SEL($\mathcal{R}_i, \mathcal{R}_j$)). Details related to the definition, efficient implementation and updates of these estimates follow.

1) Coverage: COV($\mathcal{R}_i$) estimates the progress made by the sampling-based motion planner in covering $\mathcal{R}_i$ by states in $\mathcal{T}$. Coverage estimates were introduced in the context of Monte Carlo methods as a way to measure the quality of quasirandom sampling [64]. One such measure is dispersion. As noted in [65], while dispersion has been used in sampling-based motion planning to generate quasirandom samples [66], its use as a coverage estimate is impeded by the significant cost required to compute it in high dimensions.

As an alternative to dispersion, to efficiently compute COV($\mathcal{R}_i$), SyCLoP overlays an implicit grid (denoted as *CovGrid*) over the workspace. Note that *CovGrid* is different and more fine grained than the workspace decomposition (Section IV-A1) used by the discrete layer for the computation of leads. COV($\mathcal{R}_i$) is then computed as the number of grid cells in $\mathcal{R}_i$ that contain states of vertices v from $\mathcal{T}$, i.e.,

$$\text{COV}(\mathcal{R}_i) = |\{c : c \in \textit{CovGrid} \wedge v \in \mathcal{T} \wedge \text{PROJ}(v.s) \in c \cap \mathcal{R}_i\}|.$$

Note that COV($\mathcal{R}_i$) needs to be updated only when a new vertex v is added to $\mathcal{T}$ (Algo. 1:17). To make this update efficient, each $\mathcal{R}_i$ maintains its own list of coverage cells, $\mathcal{R}_i.cells$. When v is added to $\mathcal{T}$, the list $\mathcal{R}_i.cells$ is updated accordingly. More specifically, the projection of $v.s$ onto the workspace is computed by PROJ($v.s$), which is then used to locate the region $\mathcal{R}_i$ in the workspace decomposition where PROJ($v.s$) $\in \mathcal{R}_i$. The cell c that PROJ($v.s$) belongs to is then added to the list $\mathcal{R}_i.cells$, if not already there. A hash-map is

used to efficiently search if c is already in $\mathcal{R}_i$.*cells*. In this way, $\text{COV}(\mathcal{R}_i)$ can be efficiently updated as the number of coverage cells of $\mathcal{R}_i$, i.e., $\text{COV}(\mathcal{R}_i) = |\mathcal{R}_i.\textit{cells}|$.

2) Free Volume: $\text{FREEVOL}(\mathcal{R}_i)$ provides SyCLoP with an estimate on the difficulty of exploring $\mathcal{R}_i$. The rationale is that regions that have large free volumes are easier to explore than regions that have small free volumes. SyCLoP uses sampling to estimate $\text{FREEVOL}(\mathcal{R}_i)$ as the fraction of collision-free state samples that fall into $\mathcal{R}_i$. More specifically, in a preprocessing stage, SyCLoP generates a number of samples uniformly at random from $\mathcal{S}$. For each sample s, SyCLoP computes the region $\mathcal{R}_i$ that s falls into, i.e., $\mathcal{R}_i \leftarrow \text{LOCATEREGION}(\text{PROJ}(s))$. If s is a collision-free sample, i.e., $\text{VALID}(s) = \texttt{true}$, then SyCLoP increments the number of valid samples ($n_{\textit{valid}}(\mathcal{R}_i)$) that fall into $\mathcal{R}_i$. Otherwise, SyCLoP increments the number of invalid samples ($n_{\textit{invalid}}(\mathcal{R}_i)$) that fall into $\mathcal{R}_i$. Experiments in this paper use 5000 samples. Preprocessing time is small, less than 3s on a single CPU in our setup. Then,

$$\text{FREEVOL}(\mathcal{R}_i) = \frac{\epsilon + n_{\textit{valid}}(\mathcal{R}_i)}{\epsilon + n_{\textit{valid}}(\mathcal{R}_i) + n_{\textit{invalid}}(\mathcal{R}_i)} \text{VOL}(\mathcal{R}_i),$$

where $\epsilon > 0$ is a small constant to avoid divisions by zero and $\text{VOL}(\mathcal{R}_i)$ is the volume of $\mathcal{R}_i$ in the workspace decomposition, which corresponds to the cell volume in a grid-based decomposition and triangle area in a triangulation decomposition. In this way, a region $\mathcal{R}_i$ is estimated to have a large free volume when a large fraction of random samples that fall into $\mathcal{R}_i$ are collision-free.

3) Connections: $\text{CONN}(\mathcal{R}_i, \mathcal{R}_j)$ estimates the progress the sampling-based motion planner has made in directly connecting $\mathcal{R}_i$ to $\mathcal{R}_j$. A direct connection from $\mathcal{R}_i$ to $\mathcal{R}_j$ occurs when an edge $(v, v_{\textit{new}})$ is added to $\mathcal{T}$, such that $\text{PROJ}(v.s) \in \mathcal{R}_i$ and $\text{PROJ}(v_{\textit{new}}) \in \mathcal{R}_j$. Then, $\text{CONN}(\mathcal{R}_i, \mathcal{R}_j)$ is defined as the coverage of $\mathcal{R}_j$ by states originating from direct connections of $\mathcal{R}_i$ to $\mathcal{R}_j$. The computation is similar to the coverage procedure of Section IV-C1. To make the updates efficient (Algo. 1:18), each $(\mathcal{R}_i, \mathcal{R}_j)$ maintains its own list of coverage cells, $(\mathcal{R}_i, \mathcal{R}_j).\textit{cells}$, as a hash map. In this way, $\text{CONN}(\mathcal{R}_i, \mathcal{R}_j)$ can be efficiently updated after each direct connection as the number of coverage cells of $(\mathcal{R}_i, \mathcal{R}_j)$, i.e., $\text{CONN}(\mathcal{R}_i, \mathcal{R}_j) = |(\mathcal{R}_i, \mathcal{R}_j).\textit{cells}|$.

4) Selections: $\text{SEL}(\mathcal{R}_i, \mathcal{R}_j)$ distinguishes between "empty" and "nonempty" decomposition edges. A decomposition edge $(\mathcal{R}_i, \mathcal{R}_j)$ is considered empty if $\mathcal{R}_i$, $\mathcal{R}_j$ have not yet been reached by $\mathcal{T}$. In such cases, $\text{SEL}(\mathcal{R}_i, \mathcal{R}_j)$ counts the number of times $(\mathcal{R}_i, \mathcal{R}_j)$ has been included in the current lead (Algo. 1:4). This allows the discrete search to change the empty decomposition edges that are included in future leads, giving the sampling-based motion planner a great degree of flexibility during exploration. This is especially relevant in early stages, where most of the decomposition edges are empty since $\mathcal{T}$ has yet to reach many of the regions. When $\mathcal{R}_i$ or $\mathcal{R}_j$ have been reached by $\mathcal{T}$, then $\text{SEL}(\mathcal{R}_i, \mathcal{R}_j)$ counts the number of times the sampling-based motion planner has selected tree vertices associated with $\mathcal{R}_i$ when extending $\mathcal{T}$ toward $\mathcal{R}_j$. This is to give preference in future iterations to those decomposition edges that have been explored less frequently in the past.

As evidenced by the experiments, the coverage, free volume, connection, and selection estimates allow the discrete search to compute low-cost leads that effectively guide the sampling-based motion planner. In this way, SyCLoP efficiently finds solutions even when other planners get stuck.

V. Experiments and Results

This section demonstrates that SyCLoP, by combining discrete search and sampling-based motion planning, significantly improves the computational efficiency of the underlying sampling-based motion planner in solving challenging motion-planning problems with dynamics. These significant speedups of up to two orders of magnitude were observed for all the sampling-based motion planners used in the experiments. This section also studies the impact of the workspace decomposition on the computational efficiency of SyCLoP.

A. Sampling-based Motion Planners used in the Comparisons

Various sampling-based motion planners, such as RRT [7], [8], ADDRRT [18], EST [9], [10], and SBL [14] are used in the comparisons. Let X denote one of these methods. The notation SyCLoP[X] refers to the version of SyCLoP which uses X to implement the sampling-based motion-planning in the continuous layer, i.e., functions SELECTVERTEX and EXTENDTREE are implemented as in X (see Algo. 1:9–10 and Section IV-B). Comparisons are then carried out between X and SyCLoP[X] to determine the impact of SyCLoP in improving the computational efficiency of X. Note that in all the experiments, SBL, which is a more recent version of EST, outperformed EST. Taking this into account, results in this paper include SBL but not EST. Results for both RRT and ADDRRT are included since in some cases RRT performed better and in other cases ADDRRT outperformed RRT. Then comparisons were carried out between RRT, ADDRRT, SBL and SyCLoP[RRT], SyCLoP[ADDRRT], SyCLoP[SBL].

Every effort was made to obtain efficient implementations of the various sampling-based motion planners by following the recommendations in the original research papers and suggested improvements in more recent papers and motion-planning books [11], [12]. Each time the tree is extended by a new trajectory, intermediate states along the trajectory are added to the tree as well, as suggested in [7], [8], [11], [12], [18], [50], i.e., corresponding to RRT-Connect approach. In all the implementations of EXTENDTREE, controls are selected uniformly at random. Other strategies, such as selecting the best control out of many, yielded similar results. RRT and ADDRRT implementations of SELECTVERTEX use goal bias (set to 0.05), which has been shown to improve their efficiency. Efficient nearest-neighbor packages available in OOPSMP [56], [57] were used for the nearest-neighbor computations. The distance metric was set to Euclidean distance on the state projection, i.e., $\rho(s_1, s_2) = ||\text{PROJ}(s_1), \text{PROJ}(s_2)||_2$. Other metrics, e.g., weighted combination of distances on position, orientation, and velocity components, did not work as well. SBL [14] was developed for geometric path planning. As a result, not all components of SBL, such as bi-directional tree and lazy-collision checking, can be used in motion planning with dynamics. The SBL implementation in this work is

obtained by using a single tree for the state-space exploration and using the selection strategies proposed in SBL.

B. Models of Ground and Flying Vehicles with Dynamics

Experiments use second-order dynamical models of cars, planar body thrusters, unicycles, "flying" unicycles, and high-dimensional tractor-trailers. Dynamics are modeled by a set of ordinary differential equations. The scaling factor is $1m = 0.14$ workspace units. A description of these models follow.

1) Car (adapted from [12, pp. 744]): The state $s = (x, y, \theta, v, \psi)$ consists of the position $(x, y) \in \mathbb{R}^2$ ($|x|, |y| \leq 3.75m$), orientation $\theta \in [-\pi, \pi)$, velocity v ($|v| \leq 3m/s$), and steering-wheel angle ψ ($|\psi| \leq 50°$). The car is controlled by setting the acceleration u_0 ($|u_0| \leq 1m/s^2$) and the rotational velocity of the steering-wheel angle u_1 ($|u_1| \leq 100°/s$). The equations of motions are $\dot{x} = v\cos(\theta)$; $\dot{y} = v\sin(\theta)$; $\dot{\theta} = v\tan(\psi)/L$; $\dot{v} = u_0$; $\dot{\psi} = u_1$, where $L = 0.5m$ is the distance between the front and rear axles. The body length and width are set to L and $0.5L$, respectively.

2) Planar Body with Two Thrusters (adapted from [11, pp. 406]): The state $s = (x, y, \theta, v_x, v_y, \omega)$ consists of the position $(x, y) \in \mathbb{R}^2$ ($|x|, |y| \leq 3.75m$), orientation $\theta \in [-\pi, \pi)$, translational velocity v_x along the x-axis ($|v_x| \leq 3m/s$), translational velocity v_y along the y-axis ($|v_y| \leq 3m/s$), and rotational velocity ω ($|\omega| \leq 100°/s$). The thruster controls are u_0 ($|u_0| \leq 0.5m/s^2$) and u_1 ($|u_1| \leq 0.5m/s^2$). The equations of motion are $\dot{x} = v_x$; $\dot{y} = v_y$; $\dot{\theta} = \omega$; $\dot{v_x} = u_0 * \cos(\theta) - u_1 * \sin(\theta)$; $\dot{v_y} = u_0 * \sin(\theta) + u_1 * \cos(\theta) + g$; $\dot{\omega} = -L * u_1$, where $L = 0.25m$ and the gravitational drift $g = 1m$. The body length and width are set to $2L$.

3) Unicycle (adapted from [12, pp. 743]): The state $s = (x, y, \theta, v, \omega)$ consists of the position $(x, y) \in \mathbb{R}^2$ ($|x|, |y| \leq 3.75m$), orientation $\theta \in [-\pi, \pi)$, translational velocity v ($|v| \leq 3m/s$), and rotational velocity ω ($|\omega| \leq 100°/s$). The unicycle is controlled by setting the translational u_0 ($|u_0| \leq 1m/s^2$) and rotational u_1 ($|u_1| \leq 25°/s^2$) accelerations. Equations of motion are $\dot{x} = v\cos(\theta)$; $\dot{y} = v\sin(\theta)$; $\dot{\theta} = \omega$; $\dot{v} = u_0$; $\dot{\omega} = u_1$. The body length and width are set to $0.5m$ and $0.25m$.

4) "Flying" Unicycle (adapted from [42]): In this model, the robot flies parallel to the XY-plane. To achieve this, the unicycle state is augmented as $s = (x, y, z, \theta, v, v_z, \omega)$, where $|z| \leq 3.75m$ and $|v_z| \leq 3m/s$. The additional equations of motion are $\dot{z} = v_z$ and $\dot{v_z} = u_z$, where u_z ($|u_z| \leq 1m/s^2$) is the flying control. The body length, width, and height are set to $1m$, $0.5m$, and $0.25m$, respectively. The flying-unicycle model was chosen to provide test cases for motion-planning problems with dynamics in 3D workspaces.

5) Tractor-Trailer (adapted from [12, pp. 731]): In this model, one or more trailers are attached to the tractor. The tractor is modeled as a car. The state also keeps track of the orientation θ_i of each trailer. As a result, the state for a tractor pulling N trailers has $5 + N$ variables. The equations of motions of the car are augmented with $\dot{\theta}_i = \frac{v}{d}\left(\prod_{j=1}^{i-1}\cos(\theta_{j-1} - \theta_j)\right)(\sin(\theta_{i-1}) - \sin(\theta))$, where $d = 0.15m$ is the hitch length, $\theta_0 = \theta$, and $1 \leq i \leq N$. By increasing the number of trailers, the tractor-trailer model provides challenging test cases for high-dimensional motion-planning problems with dynamics [67]. In the experiments in this paper, the number of trailers attached to the tractor is varied from 1 to 20, yielding problems with $6, \ldots, 25$ DOFs.

C. Measuring Computational Efficiency

For each motion-planning benchmark, 30 queries are generated at random as described in Sections V-D1–V-D3, and illustrated in Figs. 2–4. Each motion planner X is run on each query until the query is solved or a timeout of 700s is reached. To avoid the influence of outliers (e.g., when the motion planners fail to solve the query), the computational efficiency of a motion planner X, denoted as t_X, on a particular benchmark is measured as the median time to solve the 30 random queries generated for the benchmark.

In all our experiments, the projection function PROJ : $\mathcal{S} \rightarrow \mathcal{W}$ was defined as PROJ$(s) = (x, y)$, where (x, y) is the position component of the state s. In the case of the tractor-trailer models, (x, y) is the position component of the car which pulls the trailers. Note that the projection does not change when the number of trailers is increased.

Experiments were run on Rice Cray XD1 PBS and ADA clusters, where each processor runs at 2.2GHz and has up to 8GB RAM. Each run uses a single processor and a single thread, i.e., no parallelism.

D. Computational Efficiency of SyCLoP

The results presented in Section V-D are obtained by using a uniform grid decomposition of the workspace, where the grid is divided equally into 32 parts along each dimension. As discussed in Section IV-C1, the coverage grid *CovGrid* is set to a finer resolution (512 parts along each dimension) to allow for good estimates. Additional experiments with various grid and triangular decompositions are presented in Section V-E.

1) Computational Efficiency on Motion-Planning Problems with Dynamical Models of Ground Vehicles:

Fig. 2 summarizes the results of the experiments with several second-order models of ground vehicles, such as unicycles, thrusters, and cars (Section V-B). The benchmarks are chosen to vary in difficulty; the unicycle benchmark is the easiest to solve. Random queries are created by placing the robot in its initial configuration near the bottom (resp., top) of the workspace and requiring it to pass through the obstacles and reach the top (resp., bottom) of the workspace. The other state values, e.g., velocity, are set to zero. Fig. 2 illustrates some typical queries and solution trajectories. As shown in Fig. 2, SyCLoP[RRT], SyCLoP[ADDRRT], SyCLoP[SBL] obtain significant computational speedups of one order of magnitude over RRT, ADDRRT, SBL. Moreover, the speedups become more pronounced (up to two orders of magnitude) as harder problems are considered in Section V-D2 and V-D3.

2) Computational Efficiency on Motion-Planning Problems with Dynamical Models of Flying Vehicles:

The objective of these experiments is to test the computational efficiency of SyCLoP when the workspace is three-dimensional. The robot used in the experiments consists of a second-order dynamical model of a flying unicycle, as

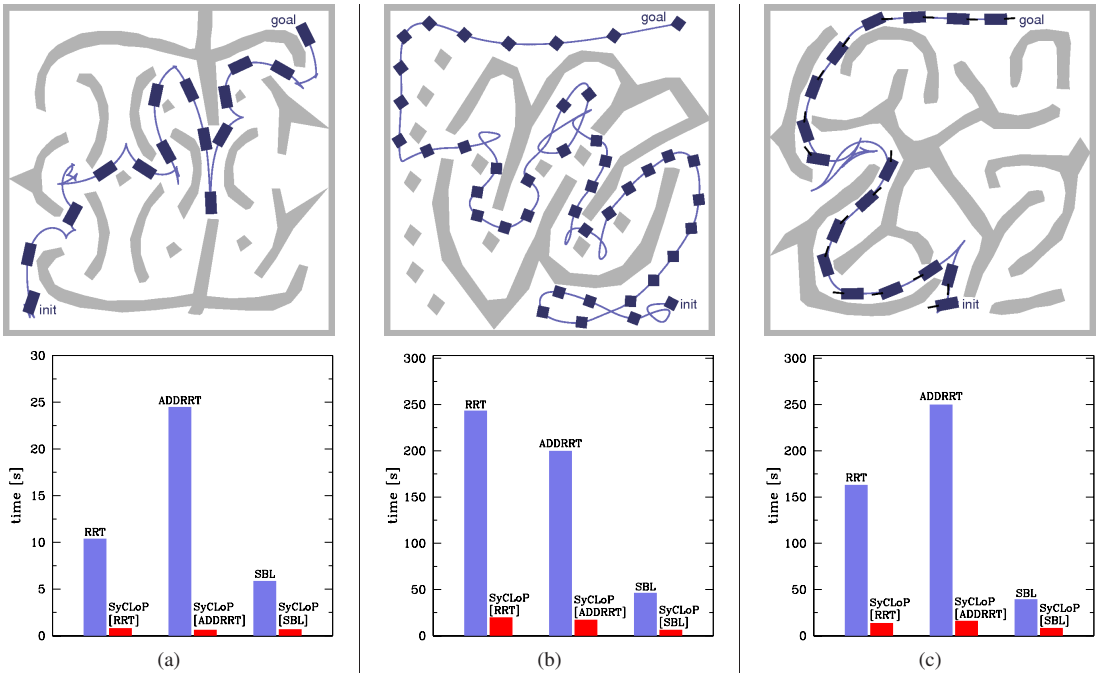


Fig. 2. Results of the experiments with second-order dynamical models of ground vehicles: (a) unicycle, (b) thruster, (c) car. The top portion of each column illustrates the workspace (shown in light gray) and a solution trajectory to a typical query (trajectory shown in dark color as an (x, y) curve together with several intermediate robot placements. In the case of the car, the steering angle is also shown. Other state values, e.g., velocity, are not shown.) The bottom portion indicates the computational efficiency of each method is measured as the median computational time in solving 30 queries.

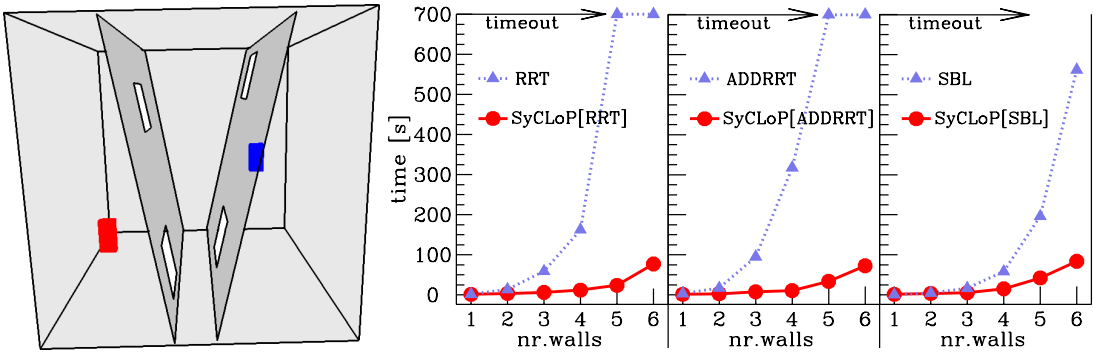


Fig. 3. Results of the experiments with a second-order dynamical model of a flying unicycle. (left) An example of the workspace with two walls and two small holes. A typical random query (initial and goal placements shown in blue and red) is also shown on the same figure. (right) Results of the experiments when the number of holes per wall is set to 2 and the number of walls is varied from 1 to 6. The computational efficiency of each method is measured as the median computational time in solving 30 queries. The maximum running time for each query is set to 700s.

described in Section V-B. Preliminary work [25], [35] did not contain such experiments. The three-dimensional workspace consists of several walls placed consecutively at a distance from each other. The walls are tilted at random angles (varying from $-10°$ to $10°$). In each wall, nonoverlapping small holes are placed at random positions. The opening of each hole is selected uniformly at random from $[1.25w, 3.0w]$, where w is the body width of the flying unicycle. This way, as suggested in [68], provides several options of varying difficulty to pass from one side of the wall to the other. Fig. 3 provides an illustration of two walls with two small randomly-placed holes. Random queries are created by placing the robot in front of the first wall and behind the last wall. In this way, a solution trajectory requires the robot to fly through the holes, passing all the walls one after the other. In the experiments, the number of walls is varied from 1 to 6 and the number of holes h for each wall is varied from 1 to 4. Fig. 3 contains a summary of the results when $h = 2$. Similar results are obtained for $h = 1, 3, 4$ (not shown due to space limitations).

Results in Fig. 3 indicate that SyCLoP significantly improves the computational efficiency of the underlying sampling-based planner. Moreover, the speedups become more pronounced as the number of walls is increased. In fact, RRT and ADDRRT time out ($t_{\text{RRT}}, t_{\text{ADDRRT}} \geq 700$s) at instances with 6 walls. Even though SBL does not time out, it still has a high computational cost: $t_{\text{SBL}} = 562.10$s. In contrast, SyCLoP efficiently solves such problems, i.e., $t_{\text{SyCLoP[RRT]}} = 77.07$s, $t_{\text{SyCLoP[ADDRRT]}} = 72.39$s, and $t_{\text{SyCLoP[SBL]}} = 83.86$s.

3) Computational Efficiency on Motion-Planning Problems with High-Dimensional Dynamical Models:

The robot consists of a second-order dynamical model of a tractor-trailer (Section V-B). By adding more trailers to

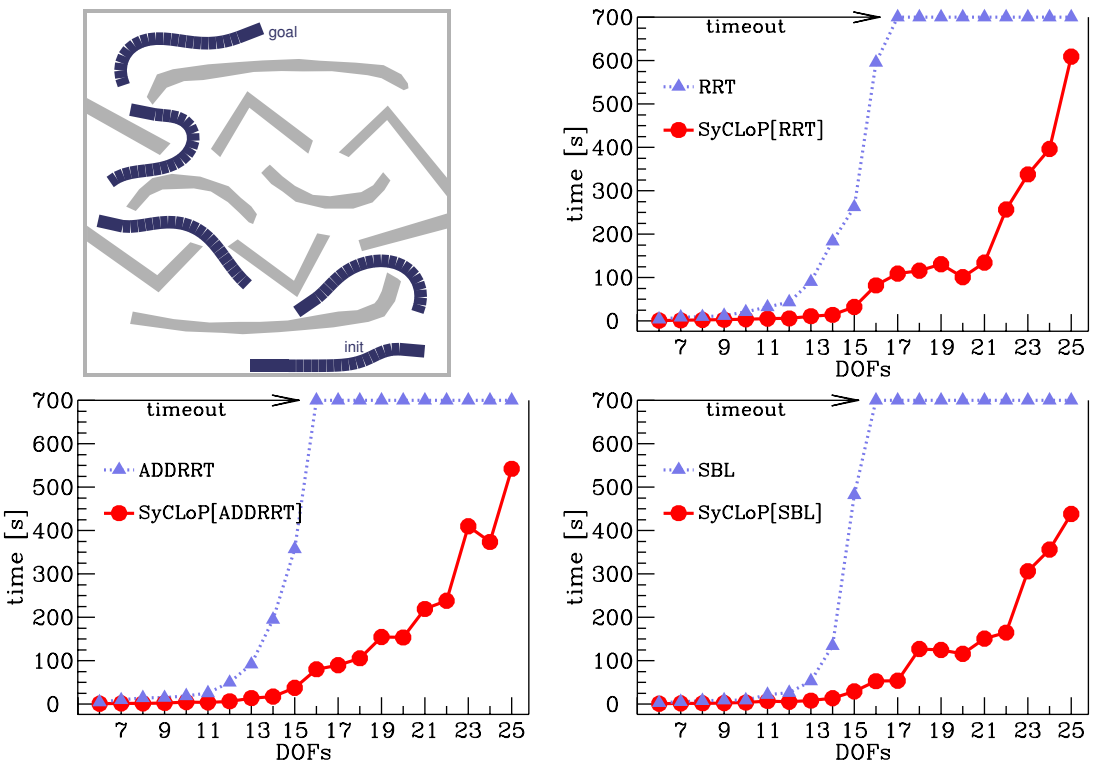


Fig. 4. Results of the experiments with a second-order dynamical model of a tractor-trailer. (left) A typical workspace (obstacles shown in light gray) and a solution trajectory to a typical query (Trajectory shown in dark as several intermediate robot configurations. Other state values, e.g., velocity, are not shown.) The illustration corresponds to a tractor pulling 20 trailers, a 25 DOF problem. (right & bottom row) Computational efficiency of each method as a function of the number of DOFs is measured as the median computational time in solving 30 queries. The maximum running time for each query is set to 700s.

increase DOFs, the tractor-trailer provides a challenging high-dimensional problem [67]. Recall that number of DOFs is $5+N$, where N is the number of trailers. In the experiments, the number of trailers varies from 1 to 20, yielding problems with $6, \ldots, 25$ DOFs. Note that preliminary work [25], [35] did not contain experiments with high-dimensional models.

As shown in Fig. 4, SyCLoP significantly improves the computational efficiency of the underlying sampling-based motion planner. As an example, on instances with 15 DOFs, $t_{\texttt{RRT}} = 262.33$s, $t_{\texttt{ADDRRT}} = 357.57$s, and $t_{\texttt{SBL}} = 482.31$s, while $t_{\texttt{SyCLoP[RRT]}} = 32.31$s, $t_{\texttt{SyCLoP[ADDRRT]}} = 37.32$s, and $t_{\texttt{SyCLoP[SBL]}} = 29.01$s, yielding speedups of 8.12, 9.58, 16.62 times, respectively. Moreover, the computational efficiency of RRT, ADDRRT, and SBL deteriorates rapidly as the number of DOFs is increased. In fact, RRT, ADDRRT, and SBL time out (set to 700s) at problems with 17 DOFs, 16 DOFs, and 16 DOFs, respectively. In contrast, SyCLoP[RRT], SyCLoP[ADDRRT], SyCLoP[SBL] solve such problems quite fast ($t_{\texttt{SyCLoP[RRT]}} = 108.89$s, $t_{\texttt{SyCLoP[ADDRRT]}} = 80.18$s, $t_{\texttt{SyCLoP[SBL]}} = 52.75$s) and efficiently handles much higher dimensional problems (for the 25 DOFs problem: $t_{\texttt{SyCLoP[RRT]}} = 609.21$s, $t_{\texttt{SyCLoP[ADDRRT]}} = 542.17$s, $t_{\texttt{SyCLoP[SBL]}} = 438.24$s).

These results demonstrate that the computational advantages of SyCLoP derive from the combination of discrete search and sampling-based motion planning. SyCLoP is shown to significantly improve the computational efficiency of the underlying sampling-based motion planner, even in cases where the original sampling-based motion planner times out.

E. Impact of Workspace Decomposition

This section provides a quantitative study of the impact of the workspace decomposition on the computational efficiency of SyCLoP. Our preliminary work [35], which carried out a similar study, used different test cases and it did not contain experiments with high-dimensional models.

Grid Decompositions: As noted, the results presented in Section V-D were obtained by using a uniform grid decomposition of the workspace, where the grid was divided equally into 32 parts along each dimension. A question that arises relates to the granularity of the grid decomposition and its impact on the computational efficiency of SyCLoP. To study this question, we repeated many of the experiments in Section V-D by using uniform grids of various granularities, i.e., grids with 1×1, 2×2, 4×4, 8×8, 16×16, 24×24, 32×32, 40×40, 48×48, 64×64, 80×80, 96×96, 112×112, 128×128, 144×144, 160×160, 176×176, 192×192, 208×208, 224×224, 240×240, and 256×256 cells.

Triangular Decompositions: In addition to grid decompositions, triangulations have been widely used. As in the study of grid decompositions, we repeated many of the experiments in Section V-D by using triangulations of various granularities. Such triangulations were obtained by setting the maximum area of each triangle in triangulation T_N to 0.000637755×2^N, and then varying $N = 0, 1, \ldots, 14$. The Triangle [69] package was used for the computations.

Fig. 5 shows a summary of the results obtained for the motion-planning problem with the second-order dynamical

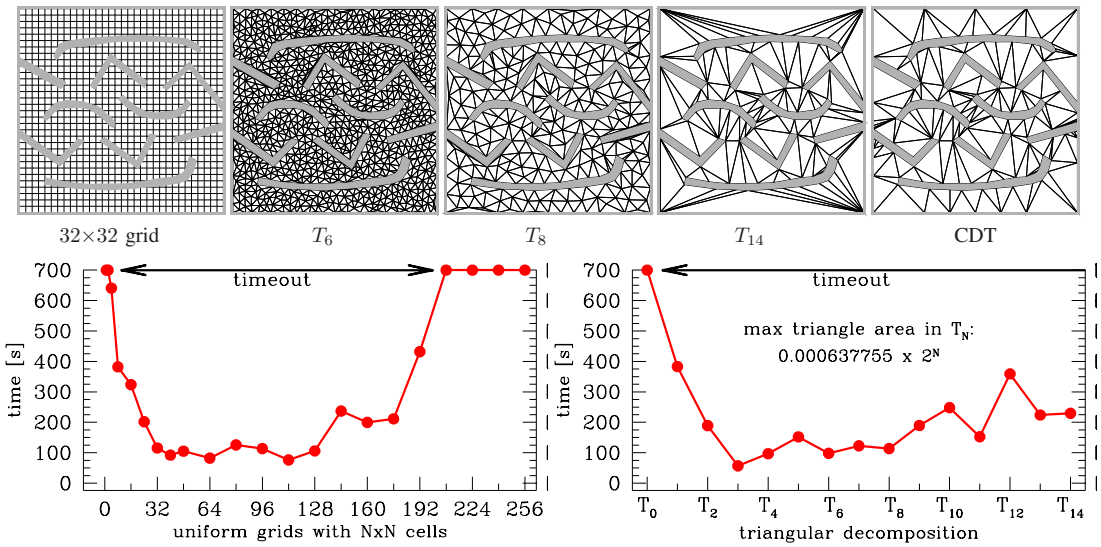


Fig. 5. Computational efficiency of SyCLoP as a function of the decomposition granularity. (top) An illustration of the 32×32 grid decomposition and several triangulations. (bottom) Results are shown for the motion-planning benchmark with the tractor-trailer dynamical model (see Section V-D3), where 15 trailers are attached to the tractor (corresponding to 20 DOFs).

model of a tractor-trailer (Section V-D3) with 15 trailers attached to the tractor (20 DOFs). Fig. 5 shows that the decomposition granularity directly impacts the computational efficiency of SyCLoP. SyCLoP is faster when the decomposition is neither too fine- nor too-coarse grained. Although finding the optimal granularity can require extensive fine-tuning, Fig. 5 shows that SyCLoP is significantly efficient for a wide range of grid and triangular decompositions. Similar trends were observed for all the motion-planning problems of Section V-D (not shown here due to space limitations). Similar observations were also made in our earlier work [35], which used a different set of motion-planning examples.

As an alternative to finding the optimal decomposition granularity, we considered conforming Delaunay triangulations for the workspace decomposition. Conforming Delaunay Triangulations (CDTs) have been widely used in computational geometry. A CDT for an environment with obstacles is similar to a Delaunay triangulation for a set of points, which maximizes the minimum angle among all possible triangulations, but could potentially differ in some places in order for the CDT to take into account polygonal edge constraints (which is handled by adding additional vertices) [69], [70].

Table I summarizes results obtained by SyCLoP when using CDTs in comparison to SyCLoP when using other triangular or grid decompositions. As mentioned earlier, lower overall computational time can be generally achieved by SyCLoP when fine-tuning the grid or the triangular decompositions (see Fig. 5 for an illustration). Finding the optimal decomposition, however, can require extensive fine-tuning. As shown in Table I, the advantage of using CDT is that it provides workspace decompositions that require no fine-tuning and allow SyCLoP to obtain close to the best computational efficiency achieved by fine-tuning other decompositions. A feature of CDT is that it produces triangulations that considerably eliminate narrow angles. This makes it easier for the sampling-based motion planner to cover the triangle during exploration. Moreover, unlike grids, CDT provides leads that avoid workspace obstacles, which further facilitates explorations. In this way, CDT provides an alternative to fine-tuning grid or triangular decompositions, which is particularly useful when applying SyCLoP to new motion-planning problems.

TABLE I
COMPUTATIONAL EFFICIENCY OF SyCLoP WHEN USING CONFORMING DELAUNAY TRIANGULATIONS (CDT) COMPARED TO SyCLoP WHEN USING OTHER TRIANGULAR AND GRID DECOMPOSITIONS

	$t_{\text{CDT}}/t_{\text{best-tri}}$	$t_{\text{CDT}}/t_{\text{best-grid}}$
unicycle: Fig. 2(a)	2.10 [T_7]	2.21 [$G_{64\times64}$]
thruster: Fig. 2(b)	1.88 [T_4]	0.99 [$G_{64\times64}$]
car: Fig. 2(c)	2.95 [T_8]	1.99 [$G_{32\times32}$]
tractor-trailer: Fig. 4		
14 DOFs	2.00 [T_6]	2.19 [$G_{128x128}$]
16 DOFs	1.29 [T_7]	1.02 [$G_{64\times64}$]
18 DOFs	2.78 [T_9]	1.48 [$G_{32\times32}$]
20 DOFs	4.79 [T_3]	3.60 [$G_{112\times112}$]
22 DOFs	1.47 [T_5]	1.53 [$G_{32\times32}$]
24 DOFs	1.74 [T_8]	1.39 [$G_{48\times48}$]

t_{CDT} denotes the computational efficiency of SyCLoP when using conforming Delaunay triangulation. $t_{\text{best-grid}}$ denotes the best computational efficiency of SyCLoP when using one of the grid decompositions. $t_{\text{best-tri}}$ denotes the best computational efficiency of SyCLoP when using one of the triangular decompositions. Decomposition that achieves the best computational efficiency is denoted inside brackets. Results correspond to SyCLoP[SBL]. Similar results were obtained for other versions of SyCLoP.

VI. DISCUSSION

To effectively solve challenging motion-planning problems with dynamics, this paper developed a multi-layered approach, SyCLoP, that synergistically combines discrete search and sampling-based motion planning. Discrete search guides the sampling-based motion planning during the tree-based exploration of the state space. Information gathered during exploration is in turn fed back from the sampling-based motion planner to the discrete search to compute increasingly low-cost leads in future iterations. In this way, leads become increasingly useful in guiding the sampling-based motion planner

toward a solution. We used several state-of-the-art motion planners, as discussed in the experiments section, to implement the sampling-based motion planning in the continuous layer of SyCLoP. Simulation experiments on high-dimensional motion-planning problems with second-order dynamical models of ground- and flying-robotic vehicles demonstrated that SyCLoP significantly improved the computational efficiency of the underlying sampling-based motion planner. More importantly, these significant speedups were observed for all the sampling-based motion planners used in the experiments and became more pronounced when considering high-dimensional motion-planning problems with dynamics. These results demonstrate the advantages of treating motion planning with dynamics not just as a search problem in a continuous space but rather as a search problem in a hybrid space consisting of discrete and continuous components.

SyCLoP opens up several venues for future research. As we consider increasingly challenging problems, it becomes important to make use of parallel or multi-threaded computational resources to significantly improve the computational efficiency of SyCLoP. Another direction for research relates to the improvement of the individual components in SyCLoP and their interplay. While this paper relied on workspace decompositions, which are motivated by their suitability for problems in mobile robotics and their low dimensionality, other types of decompositions may further improve SyCLoP.

ACKNOWLEDGMENT

The authors thank the reviewers for their comments. This work has been supported by NSF CNS 0615328 (EP, LK, MV), NSF IIS 0713623 (LK), a Sloan Fellowship (LK), NSF CCF 0613889 (MV), and developed on equipment supported by NSF CNS 0454333 and NSF CNS 0421109 in partnership with Rice University, AMD, and Cray.

REFERENCES

[1] J. Reif, "Complexity of the mover's problem and generalizations," in *IEEE Symp. Found. Comp. Sci.*, San Juan, Puerto Rico, 1979, pp. 421–427.

[2] J. T. Schwartz and M. Sharir, "On the piano movers' problem: II. General techniques for computing topological properties of algebraic manifolds," *Comm. on Pure and Appl. Math.*, vol. 36, pp. 345–398, 1983.

[3] R. Brooks and T. Lozano-Pérez, "A subdivision algorithm in configuration space for findpath with rotation," *IEEE Trans. Syst., Man, Cybern.*, vol. 15, pp. 224–233, 1985.

[4] J. T. Schwartz and M. Sharir, "A survey of motion planning and related geometric algorithms," *Artificial Intelligence*, vol. 37, pp. 157 – 169, 1988.

[5] J. Canny, *The Complexity of Robot Motion Planning*. Cambridge, MA: MIT Press, 1988.

[6] 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. Autom.*, vol. 12, no. 4, pp. 566–580, 1996.

[7] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Computer Science Department, Iowa State University, Ames, Iowa, Tech. Rep. 98-11, 1998.

[8] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *Int. J. Robot. Res.*, vol. 20, no. 5, pp. 378–400, 2001.

[9] D. Hsu, J. C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," in *IEEE Int. Conf. Robot. Autom.*, Albuquerque, NM, 1997, pp. 2719–2726.

[10] D. Hsu, R. Kindel, J. C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *Int. J. Robot. Res.*, vol. 21, no. 3, pp. 233–255, 2002.

[11] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.

[12] S. M. LaValle, *Planning Algorithms*. Cambridge, MA: Cambridge University Press, 2006.

[13] H. Keller, *Numerical Methods for Two-Point Boundary-Value Problems*. New York, NY: Dover, 1992.

[14] G. Sánchez and J. C. Latombe, "On delaying collision checking in PRM planning: Application to multi-robot coordination," *Int. J. Robot. Res.*, vol. 21, no. 1, pp. 5–26, 2002.

[15] K. I. Tsianos, I. A. Sucan, and L. E. Kavraki, "Sampling-based robot motion planning: Towards realistic applications," *Comp. Sci. Rev.*, vol. 1, pp. 2–11, 2007.

[16] A. Shkolnik, M. Walter, and R. Tedrake, "Reachability-guided sampling for planning under differential constraints," in *IEEE Int. Conf. Robot. Autom.*, Kobe, Japan, 2009, pp. 2859–2865.

[17] P. Cheng, E. Frazzoli, and S. LaValle, "Improving the performance of sampling-based motion planning with symmetry-based gap reduction," *IEEE Trans. Robot.*, vol. 24, no. 2, pp. 488–494, 2008.

[18] L. Jaillet, A. Yershova, S. M. LaValle, and T. Simeon, "Adaptive tuning of the sampling domain for dynamic-domain RRTs," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, Edmonton, Canada, 2005, pp. 4086–4091.

[19] L. Jaillet, J. Cortes, and T. Simeon, "Transition-based RRT for path planning in continuous cost spaces," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, Nice, France, 2008, pp. 2145–2150.

[20] S. Dalibard and J.-P. Laumond, "Control of probabilistic diffusion in motion planning," in *Int. Work. Algo. Found. Robot.*, Guanajuato, Mexico, 2008, pp. 467–481.

[21] J. Bruce and M. Veloso, "Real-time multi-robot motion planning with safe dynamics," *Multi-Robot Systems: From Swarms to Intelligent Automata*, vol. 3, pp. 159–170, 2005.

[22] N. Chakraborty, S. Akella, and J. Trinkle, "Complementarity-based dynamic simulation for kinodynamic motion planning," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, St. Louis, MO, 2009, pp. 787–794.

[23] D. Berenson, S. Srinivasa, D. Ferguson, A. C. Romea, and J. Kuffner, "Manipulation planning with workspace goal regions," in *IEEE Int. Conf. Robot. Autom.*, Kobe, Japan, 2009, pp. 618–624.

[24] H. H. Gonzalez-Banos, D. Hsu, and J. C. Latombe, *Motion planning: Recent developments*, ser. Autonomous Mobile Robots: Sensing, Control, Decision-Making and Applications, S. Ge and F. Lewis, Eds. CRC Press, 2006.

[25] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Discrete search leading continuous exploration for kinodynamic motion planning," in *Robotics: Sci. and Systems*, Atlanta, GA, 2007, pp. 326–333.

[26] A. M. Ladd and L. E. Kavraki, "Motion planning in the presence of drift, underactuation and discrete system changes," in *Robotics: Sci. and Systems*, Boston, MA, 2005, pp. 233–241.

[27] R. Alami, J. Laumond, and T. Siméon, "Two manipulation planning algorithms," in *Int. Work. Algo. Found. Robot.*, Stanford, CA, 1995, pp. 109–125.

[28] C. Nielsen and L. E. Kavraki, "A two level fuzzy PRM for manipulation planning," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, Japan, 2000, pp. 1716–1722.

[29] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Hybrid systems: From verification to falsification by combining motion planning and discrete search," *Formal Methods in System Design*, vol. 34, no. 2, pp. 157–182, 2009.

[30] T. Bretl, "Motion planning of multi-limbed robots subject to equilibrium constraints: The free-climbing robot problem," *Int. J. Robot. Res.*, no. 4, pp. 317–342, 2006.

[31] K. Hauser, T. Bretl, J. C. Latombe, K. Harada, and B. Wilcox, "Motion planning for legged robots on varied terrain," *Int. J. Robot. Res.*, vol. 27, no. 11-12, pp. 1325–1349, 2008.

[32] K. Hauser and J. C. Latombe, "Multi-modal motion planning in non-expansive spaces," in *Int. Work. Algo. Found. Robot.*, Guanajuato, Mexico, 2008, pp. 615–630.

[33] M. Stilman and J. Kuffner, "Planning among movable obstacles with artificial constraints," *Int. J. Robot. Res.*, vol. 27, no. 12, pp. 1295–1307, 2008.

[34] D. Nieuwenhuisen, A. van der Stappen, and M. Overmars, "An effective framework for path planning amidst movable obstacles," in *Int. Work. Algo. Found. Robot.*, ser. Springer Tracts in Advanced Robotics, New York, NY, 2006, vol. 47, pp. 87–102.

[35] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Impact of workspace decompositions on discrete search leading continuous exploration (DSLX) motion planning," in *IEEE Int. Conf. Robot. Autom.*, Pasadena, CA, 2008, pp. 3751–3756.

[36] R. Bohlin, "Path planning in practice: Lazy evaluation on a multi-resolution grid," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2001, pp. 49–54.

[37] J. P. van den Berg and M. H. Overmars, "Using workspace information as a guide to non-uniform sampling in probabilistic roadmap planners," *Int. J. Robot. Res.*, vol. 24, no. 12, pp. 1055–1071, 2005.

[38] O. Brock and L. E. Kavraki, "Decomposition-based motion planning: a framework for real-time motion planning in high-dimensional configuration spaces," in *IEEE Int. Conf. Robot. Autom.*, Seoul, Korea, 2001, pp. 1469–1474.

[39] R.-P. Berretty, M. H. Overmars, and A. F. van der Stappen, "Dynamic motion planning in low obstacle density environments," *Computational Geometry: Theory and Applications*, vol. 11, no. 3, pp. 157–173, 1998.

[40] H. Kurniawati and D. Hsu, "Workspace-based connectivity oracle: An adaptive sampling strategy for PRM planning," in *Int. Work. Algo. Found. Robot.*, ser. Springer Tracts in Advanced Robotics, New York, NY, 2006, vol. 47, pp. 35–51.

[41] S. Rodriguez, S. Thomas, R. Pearce, and N. Amato, "RESAMPL: A Region-Sensitive Adaptive Motion Planner," in *Int. Work. Algo. Found. Robot.*, ser. Springer Tracts in Advanced Robotics, 2006, vol. 47, pp. 285–300.

[42] A. M. Ladd and L. E. Kavraki, "Fast tree-based exploration of state space for robots with dynamics," in *Int. Work. Algo. Found. Robot.*, Utrecht/Zeist, Netherlands, 2004, pp. 297–312.

[43] F. Lingelbach, "Path planning using probabilistic cell decomposition," in *IEEE Int. Conf. Robot. Autom.*, Orlando, FL, 2004, pp. 467–472.

[44] K. E. Bekris and L. E. Kavraki, "Greedy but safe replanning under kinodynamic constraints," in *IEEE Int. Conf. Robot. Autom.*, Rome, Italy, 2007, pp. 704–710.

[45] I. Şucan and L. E. Kavraki, "On the performance of random linear projections for sampling-based motion planning," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, St. Louis, MO, 2009, pp. 2434–2439.

[46] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, 2005.

[47] P. Tompkins, A. Stentz, and D. Wettergreen, "Mission-level path planning and re-planning for rover exploration," *Robotics and Autonomous Systems*, vol. 54, no. 1, pp. 174 – 183, 2006.

[48] A. Yahja, S. Singh, and A. T. Stentz, "An efficient on-line path planner for outdoor mobile robots operating in vast environments," *Robotics and Autonomous Systems*, vol. 33, no. 1, pp. 129–143, 2000.

[49] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *IEEE Int. Conf. Robot. Autom.*, San Diego, CA, 1994, pp. 3310–3317.

[50] B. Burns and O. Brock, "Single-query motion planning with utility-guided random trees," in *IEEE Int. Conf. Robot. Autom.*, Rome, Italy, 2007, pp. 3307–3312.

[51] L. Zhang and D. Manocha, "An efficient retraction-based RRT planner," in *IEEE Int. Conf. Robot. Autom.*, Pasadena, CA, 2008, pp. 3743–3750.

[52] A. Shkolnik and R. Tedrake, "Path planning in 1000+ dimensions using a task-space Voronoi bias," in *IEEE Int. Conf. Robot. Autom.*, Kobe, Japan, 2009, pp. 2061–2067.

[53] R. Diankov, N. Ratliff, D. Ferguson, S. Srinivasa, and J. J. Kuffner, "BiSpace planning: Concurrent multi-space exploration," in *Robot.: Sci. Syst.*, Zurich, Switzerland, 2008, pp. 159–166.

[54] E. Plaku and L. E. Kavraki, "Quantitative analysis of nearest-neighbors search in high-dimensional sampling-based motion planning," in *Int. Work. Algo. Found. Robot.*, ser. Springer Tracts in Advanced Robotics, New York, NY, 2006, vol. 47, pp. 3–18.

[55] A. Yershova and S. M. LaValle, "Improving motion planning algorithms by efficient nearest-neighbor searching," *IEEE Trans. Robot.*, vol. 23, no. 1, pp. 151–157, 2007.

[56] E. Plaku, K. E. Bekris, and L. E. Kavraki, "OOPS for Motion Planning: An Online Open-source Programming System," in *IEEE Int. Conf. Robot. Autom.*, Rome, Italy, 2007, pp. 3711–3716.

[57] E. Plaku and L. E. Kavraki, "OOPSMP," 2007. [Online]. Available: http://www.kavrakilab.org

[58] A. Yershova, "MPNN: Nearest neighbor library for motion planning." [Online]. Available: http://www.cs.duke.edu/∼yershova/software/MPNN

[59] E. Frazzoli, M. A. Dahleh, and E. Feron, "Maneuver-based motion planning for nonlinear systems with symmetries," *IEEE Trans. Robot.*, vol. 21, no. 6, pp. 1077–1091, 2005.

[60] M. S. Branicky, R. A. Knepper, and J. J. Kuffner, "Path and trajectory diversity: Theory and algorithms," in *IEEE Int. Conf. Robot. Autom.*, Pasadena, CA, 2008, pp. 1359–1364.

[61] L. H. Erickson and S. M. LaValle, "Survivability: Measuring and ensuring path diversity," in *IEEE Int. Conf. Robot. Autom.*, Kobe, Japan, 2009, pp. 2068–2073.

[62] M. de Berg, O. Cheong, M. van Kreveld, and M. H. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed. Springer-Verlag, 2008.

[63] R. Geraerts and M. Overmars, "A comparative study of probabilistic roadmap planners," in *Int. Work. Algo. Found. Robot.*, Nice, France, 2002, pp. 43–58.

[64] M. Drmota and R. F. Tichy, *Sequences, discrepancies and applications*. Springer, Berlin, 1997.

[65] J. Esposito, J. Kim, and V. Kumar, "Adaptive RRTs for validating hybrid robotic control systems," in *Int. Work. Algo. Found. Robot.*, Zeist, Netherlands, 2004, pp. 107–132.

[66] S. LaValle, M. Branicky, and S. Lindemann, "On the relationship between classical grid search and probabilistic roadmaps," *Int. Journal of Robotics Research*, vol. 23, no. 7–8, pp. 673–692, 2003.

[67] J. Laumond, "Controllability of a multibody mobile robot," *IEEE Trans. Robot. Autom.*, vol. 9, no. 6, pp. 755–763, 1993.

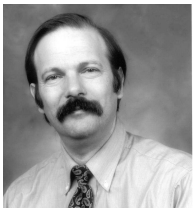
[68] E. Plaku, K. E. Bekris, B. Y. Chen, A. M. Ladd, and L. E. Kavraki, "Sampling-based roadmap of trees for parallel motion planning," *IEEE Trans. Robot.*, vol. 21, no. 4, pp. 597–608, 2005.

[69] J. R. Shewchuk, "Delaunay refinement algorithms for triangular mesh generation," *Computational Geometry: Theory and Applications*, vol. 22, no. 1-3, pp. 21–74, 2002.

[70] ——, "General-dimensional constrained delaunay and constrained regular triangulations, i: Combinatorial properties," *Discrete & Computational Geometry*, vol. 39, pp. 580–637, 2008.

Erion Plaku is a Postdoctoral Fellow at the Laboratory for Computational Sensing and Robotics in the Department of Computer Science at Johns Hopkins University. The work in this paper was conducted while he was a Ph.D. student at Rice University. He received the Ph.D. degree in Computer Science from Rice University in 2008. His research focuses on motion planning and discrete planning for automatic or human-machine cooperative task performance in complex domains, such as mobile robotics, manipulation, hybrid systems, and robotic-assisted surgery.

Lydia E. Kavraki Lydia E. Kavraki is the Noah Harding Professor of Computer Science and Bioengineering at Rice University. Her research interests are in physical algorithms and their applications in robotics and biology. Kavraki was a co-inventor of the Probabilistic RoadMap Planner (PRM). She is the author/co-author of more than 150 technical papers and a robotics textbook. Kavraki is the recipient of the ACM Grace Murray Hopper Award and the Early Academic Career Award from the IEEE RAS Society. She is also a fellow of AAAI and AIMBE. Information about her work can be found at www.kavrakilab.org



Moshe Y. Vardi is the George Professor in Computational Engineering and Director of the Ken Kennedy Institute for Information Technology Institute at Rice University. He is the co-recipient of three IBM Outstanding Innovation Awards, the ACM SIGACT Goedel Prize, the ACM Kanellakis Award, the ACM SIGMOD Codd Award, and the Blaise Pascal Medal. He is the author and co-author of over 350 technical papers, as well as two books: "Reasoning about Knowledge" and "Finite Model Theory and Its Applications". He is a Fellow of IEEE, ACM, AAAI, and AAAS. He is a member of the US National Academy of Engineering, the European Academy of Science, and Academia Europea.