IEEE Transactions on Robotics, 26(3):469–482, 2010. DOI: 10.1109/TRO.2010.2047820

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

Erion Plaku Lydia E. Kavraki Moshe Y. Vardi

***Abstract*—Effectively incorporating robot dynamics into motion planning has been an active area of research in robotics over the last decade. Toward this goal, this work proposes a novel multi-layered approach, termed Synergistic Combination of Layers of Planning (SyCLoP), that synergistically combines high-level discrete planning and sampling-based motion planning.**

Initially, SyCLoP uses a workspace decomposition to construct a discrete model of the motion-planning problem. At each iteration, high-level planning, which draws from research in AI and logic, searches the discrete model for a feasible plan, i.e., a sequence of decomposition regions that can effectively guide motion planning as it extends a tree during the search for a solution trajectory. In return, information gathered by motion planning, such as progress made in connecting decomposition regions, is fed back to high-level planning. In this way, the planning layers in SyCLoP are not independent, but work in tandem to compute in future iterations increasingly feasible high-level plans and quickly grow the tree toward the goal.

The synergistic combination of high-level discrete planning and sampling-based motion planning allows SyCLoP to effectively solve challenging motion-planning problems with dynamics. Simulation experiments with high-dimensional dynamical models of ground and flying vehicles demonstrate computational speedups of up to two orders of magnitude over state-of-the-art motion planners. In addition, SyCLoP is well-suited for hybrid systems, which move beyond continuous models by employing discrete logic to instantaneously modify the dynamics.

I. Introduction

A basic component of robot autonomy is the ability of the robot to plan the motions needed to reach a goal destination while avoiding collisions. In its early years, research in motion planning considered only geometric models of the robot and the environment on which the robot operates [1]–[3]. This geometric setting served to fuel research in sampling-based motion planning, giving rise to popular methods such as the Probabilistic RoadMap (PRM) [4], Rapidly-exploring Random Tree (RRT) [5], [6], Expansive Space Tree (EST[1]) [7], [8], and many others, as surveyed in [9]–[12].

The success of sampling-based approaches and the need to produce in simulations solutions that respect physical constraints in robot motion led researchers to go beyond geometric models and to incorporate robot dynamics directly into motion planning [5]–[8], [13]–[27] (see [9]–[12] for more overviews). Sampling-based methods, such as RRT [5], [6] and EST [7], [8] have now been widely used in motion planning with dynamics. These motion planners typically explore the state space by maintaining a tree data structure, which is extended with trajectories obtained by applying controls and simulating the dynamics. This greatly facilitates the design of controllers, which are needed in order to generate hardware commands that can enable the robot to follow in the physical world solutions obtained in simulation [28].

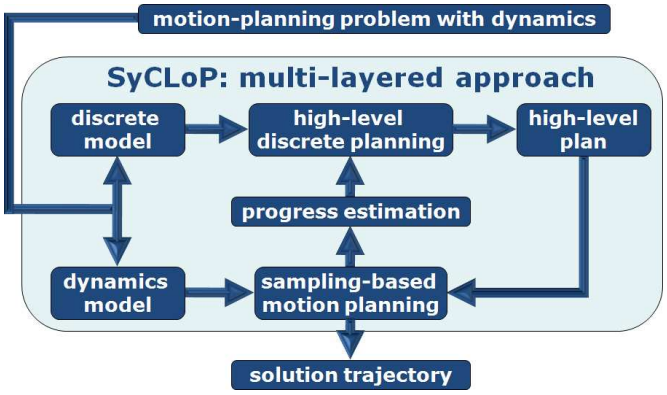

Fig. 1. Proposed multi-layered approach, SyCLoP, synergistically combines high-level discrete planning and sampling-based motion planning.

While significant progress has been made, motion planning with dynamics still constitutes a significant challenge. Taking into account the robot dynamics in addition to its geometry can considerably increase the dimensionality and the computational complexity of the motion-planning problem [29]–[32].

The contribution of this paper is a novel multi-layered approach to motion planning with dynamics, termed Synergistic Combination of Layers of Planning (SyCLoP), that synergistically combines high-level discrete planning and sampling-based motion planning. SyCLoP draws from our preliminary work [23], [24], further improves the proposed method, and contains new simulation experiments on motion-planning problems with high-dimensional dynamical models of ground and flying vehicles. SyCLoP, as other sampling- and tree-based motion planners, uses a tree exploration of the state space. As demonstrated in this paper, however, as a result of the synergistic combination of high-level discrete planning and sampling-based motion planning, SyCLoP is considerably faster (up to two orders of magnitude) than state-of-the-art motion planners in solving challenging motion-planning problems with dynamics.

Erion Plaku, Lydia E. Kavraki, and Moshe Y. Vardi are with the Department 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 method, and presents results on additional motion-planning problems involving high-dimensional second-order dynamical models of ground and flying vehicles.

Manuscript received April 3, 2009.

[1]The acronym EST does not appear in the original papers.

In motion planning with dynamics, tree-based approaches have become the norm. Over the years, numerous strategies have been proposed to effectively guide the tree exploration of the state space. RRT [5], [6] uses a distance metric and nearest neighbors to introduce a Voronoi bias to the tree exploration. EST [6], [7] maintains a density distribution over the states in the tree to guide the exploration toward areas of low density. The work in [33] defines the utility of each state in an information-theoretic sense and extends the tree from those states that would increase the overall utility. The work in [34] extends the tree using a framework that aims to balance exploration with exploitation of the configuration space. Other tree-based motion planners use decompositions to guide the exploration. In fact, the idea of using decompositions appeared early in motion-planning literature. Key theoretical results and some of the first motion planners were obtained using decompositions, cf. [1], [2], [9], [35], [36]. When sampling-based motion planners became popular, the idea of decomposition-based approaches has been revisited many times. In the context of roadmap motion planners, the work in [37]–[40] uses decompositions to guide the sampling strategy during roadmap construction. In tree-based motion planning, the work in [41] uses a grid and simple selection strategies to extend the tree from less-populated cells. The work in [42] uses approximate cell decompositions of the configuration space to guide the search toward the goal. The work in [43] uses partial workspace decompositions to speed up exploration of configuration space. In motion planning with dynamics, the work in [21], [27] uses a potential field over a grid to guide the tree exploration toward the goal. The work in [18] uses a subdivision scheme to extend the tree from regions that have been selected less frequently in the past. The work in [22] also uses subdivision, but guides exploration toward regions with low coverage. We acknowledge that due to the large body of research in sampling-based motion planning, the summary in this paper, which focused on recent work, covers only a small fraction of related work. We refer the reader to recent books and surveys for more complete overviews [10]–[12].

SyCLoP relies on a combination of high-level discrete planning and sampling-based motion planning in order to effectively solve challenging motion-planning problems with dynamics. Fig. 1 provides a schematic representation of this combination. Initially, SyCLoP uses a workspace decomposition to construct a high-level discrete model of the motion-planning problem. The discrete model is represented in terms of a graph whose vertices are regions in the decomposition and whose edges denote the physical adjacency of the regions. Drawing from research in logic and AI [44]–[46], high-level planning in SyCLoP exploits the simple observation that any solution trajectory corresponds to some *high-level plan*, i.e., a sequence of neighboring decomposition regions that starts and ends at regions associated with the initial and goal states, respectively. Although the converse does not hold in general, a high-level plan can, however, provide a general direction along which to extend the tree. Sampling-based motion planning then can attempt to obtain a solution trajectory by extending the tree from one region to its neighbor in the high-level plan.

In motion-planning literature, two-layered approaches have consisted of two independent layers. Examples include early planners, such as SANDROS [47], and more recent work in motion planning for physical systems. As an example, the work in [48] first obtains near-optimal high-level discrete plans via D* search and then uses controllers to closely follow the discrete plan in the physical world.

SyCLoP builds further upon this idea of a two-layered approach to motion planning with dynamics by

(i) incorporating state-of-the-art sampling-based motion planning in the second layer, and, more importantly,
(ii) instead of treating the planning layers as independent from each other, SyCLoP *synergistically* combines high-level planning and sampling-based motion planning.

More specifically, since the discrete model can provide many alternative high-level plans, as shown in Fig. 1, the planning layers in SyCLoP work in tandem to evaluate the feasibility of current plans and to compute increasingly feasible plans in future iterations. The feasibility of a high-level plan is estimated based on information gathered during motion planning, such as progress made in connecting decomposition regions, time spent in exploration, and region coverage. In order to compute the feasibility estimates efficiently, SyCLoP uses a second workspace decomposition, which is more fine grained than the workspace decomposition used for computing high-level plans. Drawing from earlier work in sampling-based motion planning [18], [21]–[27], [33], [41], earlier versions of our work [23], [24], and extensive experimentations, we note that the feasibility 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.

Aiming to strike a balance between greedy and methodical search, SyCLoP gives priority to highly feasible plans, but at the same time it does not ignore other less feasible plans. In this way, SyCLoP has the flexibility to extend the tree along feasible directions while able to radically change direction if information from motion planning suggests other more feasible plans.

The synergistic combination of high-level planning and sampling-based motion planning allows SyCLoP to effectively solve challenging motion-planning problems with dynamics. Simulation experiments with second-order dynamical models of ground and flying vehicles demonstrate computational speedups of up to two orders of magnitude over state-of-the-art motion planners. The computational efficiency of SyCLoP becomes more pronounced when considering high-dimensional motion-planning problems with dynamics. Simulation experiments in these cases show that SyCLoP remains efficient, while the computational efficiency of the other motion planners in the comparisons deteriorates rapidly as the number of degrees-of-freedom (DOFs) increases.

In addition to motion planning with dynamics, SyCLoP is well-suited for hybrid systems [49]. Hybrid systems move beyond continuous models by employing discrete logic to instantaneously modify the dynamics to respond to mishaps or unanticipated changes [50]. Hybrid systems are used in a wide variety of settings, such as in embedded controllers in the automotive industry, and also for modeling biological networks and air-traffic management systems [51], [52].

The paper is organized as follows. The motion-planning problem with dynamics and a basic tree-search framework commonly used to solve such problems are described in Section II. SyCLoP is described in Section III. Experiments and results are described in Section IV, which also includes a study on the impact of the discrete model and the interplay between high-level planning and sampling-based motion planning on the efficiency of SyCLoP. Applications of SyCLoP to motion planning for hybrid systems are discussed in Section V. The paper concludes with a discussion in Section VI.

II. Preliminaries

A. The Motion-Planning Problem with Dynamics

Robots are often controlled by applying external inputs. As an example, a car is driven by applying acceleration and rotating the steering wheel. The dynamics describe the evolution of a system's state. This section defines the motion-planning problem using a general formulation that treats the dynamics as a black box. Similar to the abstraction of collision checking in sampling-based approaches, the definition of dynamics as a black-box allows the motion planner to access the necessary components for planning purposes, while hiding the intricacies of the robot and its interactions with the environment.

Definition II-A.1. *A motion-planning problem with dynamics is a tuple* $\mathcal{P} = (\mathcal{S}, \mathcal{U}, \textsc{Flow}, \textsc{Valid}, s_{init}, \textsc{Goal})$*, 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;*
- $\textsc{Flow} : \mathcal{S} \times \mathcal{U} \times \mathbb{R}^{\geq 0} \to \mathcal{S}$ *is a flow function that simulates the system dynamics when an input is applied to the system for a certain time duration.*
- $\textsc{Valid} : \mathcal{S} \to \{\texttt{true}, \texttt{false}\}$ *specifies state constraints;*
- $s_{init} \in \mathcal{S}$ *is an initial state;*
- $\textsc{Goal} : \mathcal{S} \to \{\texttt{true}, \texttt{false}\}$ *specifies the goal.*

A solution to the motion-planning problem $\mathcal{P}$ *is a valid trajectory* $\gamma : [0, T] \to \mathcal{S}$ *that starts at* s_{init} *and satisfies the motion-planning goal, i.e.,* $\gamma(0) = s_{init}; \textsc{Goal}(\gamma(T)) = \texttt{true}$*; and* $\forall t \in [0, T] : \textsc{Valid}(\gamma(t)) = \texttt{true}$*. Moreover,* γ *is obtained by applying to* s_{init} *a sequence of input controls* $u_1, u_2, \ldots, u_N \in \mathcal{U}$ *for certain time durations* $T_1, T_2, \ldots, T_N \in \mathbb{R}^{\geq 0}$*, i.e.,*

$$\gamma = s_{init} \circ (u_1, T_1) \circ \ldots \circ (u_N, T_N),$$

where $\gamma_A \circ (u_B, T_B)$ *denotes the extension of the trajectory* $\gamma_A : [0, T_A] \to \mathcal{S}$ *by applying the input control* u_B *to the state* $\gamma_A(T_A)$ *for* T_A *time units, i.e.,*

$$(\gamma_A \circ (u_B, T_B))(t) = \begin{cases} \gamma_A(t), \text{ if } 0 \leq t \leq T_A \\ \textsc{Flow}(\gamma_A(T_A), u_B, t - T_B), \text{ otherwise} \end{cases}$$

Below we briefly discuss common representations used in sampling-based motion planning to model dynamics, state constraints, and the motion-planning goal.

1) Dynamics: When a system is at a state $s \in \mathcal{S}$ and a control $u \in \mathcal{U}$ is applied for $t \in \mathbb{R}^{\geq 0}$ time units, the system's state evolves according to the underlying dynamics and at the end the system may be at a new state $s_{new} \in \mathcal{S}$. Such behavior is captured by a flow function $\textsc{Flow} : \mathcal{S} \times \mathcal{U} \times \mathbb{R}^{\geq 0} \to \mathcal{S}$, where, for each $s \in \mathcal{S}$, $u \in \mathcal{U}$, and $t \in \mathbb{R}^{\geq 0}$, $\textsc{Flow}(s, u, t)$ outputs the new state $s_{new} \in \mathcal{S}$ obtained by applying the input u for t time units when the system is at state s. Consistency in the flow function is ensured by the following requirements:

- *(Identity)* $\forall s \in \mathcal{S}, u \in \mathcal{U}$: $s = \textsc{Flow}(s, u, 0)$.
- *(Transitivity)* $\forall s \in \mathcal{S}, u \in \mathcal{U}, t_1, t_2 \in \mathbb{R}^{\geq 0}$: $\textsc{Flow}(s, u, t_1 + t_2) = \textsc{Flow}(\textsc{Flow}(s, u, t_1), u, t_2)$.

For many systems, dynamics are commonly described by a set of differential equations $g : \mathcal{S} \times \mathcal{U} \to \dot{\mathcal{S}}$. Closed-form solutions (if available) or numerical integrations can be used to compute Flow from g. Several examples of second-order dynamics are given in Section IV.

In addition to differential equations, physics-based simulations can be used to model the dynamics. Physics-based simulations provide an increased level of realism by also modeling friction, gravity, and interactions of the robot with the environment, which cannot be easily described analytically.

2) State Constraints: State constraints indicate a desired invariant that states should satisfy, e.g., collision avoidance, bounds on velocity, turning radius. This work allows for general specifications as a function $\textsc{Valid} : \mathcal{S} \to \{\texttt{true}, \texttt{false}\}$, where $\textsc{Valid}(s) = \texttt{true}$ iff s satisfies the state constraints.

3) Motion-Planning Goal: The motion-planning goal is specified as desired constraints that a goal state should satisfy, e.g., desired configuration, velocity. As in the case of Valid, this work allows for general specifications as $\textsc{Goal} : \mathcal{S} \to \{\texttt{true}, \texttt{false}\}$, where $\textsc{Goal}(s) = \texttt{true}$ iff s satisfies the motion-planning goal. In many cases, as in this paper, Goal is defined as a small ball centered at a goal state, i.e.,

$$\textsc{Goal}(s) = \texttt{true} \text{ iff } \rho(s, s_{goal}) \leq \delta,$$

where $s_{goal} \in \mathcal{S}$ is the goal state, $\rho : \mathcal{S} \times \mathcal{S} \to \mathbb{R}^{\geq 0}$ is a distance metric, and $\delta > 0$ is a distance threshold.

B. A Basic Tree-Search Framework for Motion Planning with Dynamics in Sampling-based Approaches

Motion planning with dynamics is generally approached as a search problem for a valid trajectory $\gamma : [0, T] \to \mathcal{S}$ that satisfies the motion-planning goal. Many sampling-based motion planners follow a common framework that searches for a solution by extending in the state space $\mathcal{S}$ a tree rooted at the initial state s_{init} [5]–[8], [12], [17]–[19], [21]–[27], [33]. Pseudocode is given in Algo. 1.

A search data structure is maintained as a tree $\mathcal{T}$, where each vertex $v \in \mathcal{T}$ has a backpointer, $v.parent$, to its parent. Moreover, v is associated with a state $s \in \mathcal{S}$, written as $v.s$, and the edge $(v.parent, v)$ indicates that a valid trajectory connects $v.parent.s$ to $v.s$. A description of Algo. 1 follows.

1) InitializeTree($\mathcal{P}$)*:* During initialization, the root vertex v_{init} is associated with s_{init} and is added to $\mathcal{T}$. As the search proceeds iteratively, $\mathcal{T}$ is extended by adding new vertices.

Algorithm 1 A Basic Tree-Search Framework

Input: $\mathcal{P} = (\mathcal{S}, \mathcal{U}, \text{FLOW}, \text{VALID}, s_{init}, \text{GOAL})$
$t_{\max} \in \mathbb{R}^{>0}$: upper bound on computation time
$\epsilon \in \mathbb{R}^{>0}$: propagation step
$n_{add} \in \mathbb{N}$: nr. steps to add new vertex to tree
Output: A solution trajectory or `null`

$\mathcal{T} \leftarrow \text{INITIALIZETREE}(\mathcal{P})$ ◇II-B1
while $\text{ELAPSEDTIME} < t_{\max}$ **do**
  $v \leftarrow \text{SELECTVERTEXFROMTREE}(\mathcal{P}, \mathcal{T})$ ◇II-B2
  $\gamma \leftarrow \text{EXTENDTREE}(\mathcal{P}, \mathcal{T}, v, \epsilon, n_{add})$ ◇II-B3
  if $\gamma \neq$ `null` **then return** γ ◇*solution trajectory*
return `null`

$\text{EXTENDTREE}(\mathcal{P}, \mathcal{T}, v, \epsilon, n_{add})$
1: $[u, \max_{nsteps}] \leftarrow \text{SAMPLECONTROLANDMAXNRSTEPS}(\mathcal{P}, \mathcal{T}, v)$
2: $s_0 \leftarrow v.s$; *count* $\leftarrow 0$
3: **for** $i = 1 \ldots \max_{nsteps}$ **do**
4: $s_i \leftarrow \text{FLOW}(s_{i-1}, u, \epsilon)$; *count* $\leftarrow$ *count* $+ 1$
5: **if** $\text{VALID}(s_i) =$ `false` **then**
6: **return** `false`
7: *goal* $\leftarrow \text{GOAL}(s_i)$
8: **if** *goal* $=$ `true` **or** *count* $\geq n_{add}$ **then**
9: $v_{new}.[s, u, t, parent] \leftarrow \text{NEWVERTEX}(s_i, u,$ *count* $* \epsilon, v)$
10: $\mathcal{T} \leftarrow \mathcal{T} \cup \{v_{new}\}$; *count* $\leftarrow 0$
11: **if** *goal* $=$ `true` **then return** $\text{TRAJ}(\mathcal{T}, v_{new}.s)$

2) $\text{SELECTVERTEXFROMTREE}(\mathcal{P}, \mathcal{T})$*:* This function selects at each iteration a vertex $v \in \mathcal{T}$ from which to extend $\mathcal{T}$. Over the years, numerous strategies have been proposed that rely on distance metrics, nearest neighbors, probability distributions, and many others [10], [11]. As an example, the initial version of `RRT` [5] first samples a state $s \in \mathcal{S}$ uniformly at random and then selects $v \in \mathcal{T}$ whose $v.s$ is the closest to s according to a distance metric.

3) $\text{EXTENDTREE}(\mathcal{P}, \mathcal{T}, v, \epsilon, n_{add})$*:* This function extends $\mathcal{T}$ from v by computing a valid trajectory $\gamma : \mathbb{R}^{>0} \rightarrow \mathcal{S}$ that starts at $v.s$. A common strategy is to apply some input $u \in \mathcal{U}$ to $v.s$ and follow the system dynamics until the state-constraints are not satisfied or a maximum number of steps $\max_{nsteps}$ is exceeded [10], [11]. The input u is generally selected uniformly at random to allow subsequent calls to extend $\mathcal{T}$ along new directions. Intermediate states along the trajectory defined by the state $v.s$, input control u, and time duration, are added to $\mathcal{T}$, as suggested in [5], [6], [10], [11], [33], [53].

The implementation of EXTENDTREE relies on an iterative procedure. Let $\max_{nsteps}$ denote the maximum number of steps and let $\epsilon > 0$ denote the step size (Algo. 1:1). Initially, $s_0 = v.s$ (Algo. 1:2). At the i-th iteration, s_i is computed as $s_i = \text{FLOW}(s_{i-1}, u, \epsilon)$ (Algo. 1:4). If s_i is invalid, then the computation stops (Algo. 1:5–6). Otherwise, a check is performed to determine whether s_i satisfies the motion-planning goal (Algo. 1:7). A new vertex is added to $\mathcal{T}$ if s_i satisfies the motion-planning goal or several successful steps have been taken since the last addition to $\mathcal{T}$ (Algo. 1:8–10). If the motion-planning goal is satisfied, then the solution trajectory is computed by concatenating the trajectories associated with the tree edges connecting v_{init} to v_{new} (Algo. 1:11).

III. SyCLoP

The efficiency of the basic tree search (Section II-B) depends on the ability of the framework to quickly extend $\mathcal{T}$ toward the goal. To effectively guide the search, `SyCLoP` synergistically combines high-level discrete planning with sampling-based motion planning, as illustrated in Algo. 2.

Algorithm 2 `SyCLoP`

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

1: $\mathcal{T} \leftarrow \text{INITIALIZETREE}(\mathcal{P})$ ◇II-Ba
2: $\mathcal{D} \leftarrow \text{DISCRETEMODEL}(\mathcal{P})$ ◇III-A
3: $\text{INITESTIMATES}(\mathcal{P}, \mathcal{T}, \mathcal{D})$ ◇III-D
4: **while** $\text{ELAPSEDTIME} < t_{\max}$ **do**
5: $[\mathcal{R}_{i_j}]_{j=1}^{k} \leftarrow \text{HIGHLEVELPLANNING}(\mathcal{D})$ ◇III-C
6: $\gamma \leftarrow \text{GUIDEDEXPLORATION}(\mathcal{P}, \mathcal{T}, \mathcal{D}, [\mathcal{R}_{i_j}]_{j=1}^{k})$ ◇III-E
7: **if** $\gamma \neq$ `null` **then**
8: **return** γ ◇*solution trajectory*
9: $\text{UPDATEESTIMATES}(\mathcal{P}, \mathcal{T}, \mathcal{D})$ ◇III-D
10: **return** `null`

A. High-Level Discrete Model

The high-level discrete model provides a simplified high-level planning layer that can be used to effectively guide sampling-based motion planning. This allows `SyCLoP` to benefit from research in computer logic and AI, where high-level planning plays an important role [44]–[46]. Let $\mathcal{W}$ denote the workspace, i.e., the two- or three-dimensional environment (including the obstacles) on which the robot operates. The discrete models in this paper are based on decompositions of the workspace $\mathcal{W}$ 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

$$\forall \mathcal{R}_i, \mathcal{R}_j \in \mathcal{W} : \textit{Interior}(\mathcal{R}_i) \cap \textit{Interior}(\mathcal{R}_j) = \emptyset.$$

In order to map states to workspace decomposition regions, `SyCLoP` first uses a projection $\text{PROJ} : \mathcal{S} \rightarrow \mathcal{W}$ to map a state $s \in \mathcal{S}$ to the corresponding point in $\mathcal{W}$ by extracting the position component from s. `SyCLoP` then uses a region-locator function $\text{LOCATEREGION} : \mathcal{W} \rightarrow \{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$ to map each workspace point to the corresponding region, i.e.,

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

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

The computation of workspace decompositions is an active research area in computational geometry [54]. 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 [54]. The impact of workspace decompositions on the overall computational efficiency of `SyCLoP` is studied in Section IV-D.

The discrete model also keeps information about the regions associated with the initial and goal states of the motion-planning problem, i.e., $R_{init} = \text{LOCATEREGION}(\text{PROJ}(s_{init}))$ and $R_{goal} = \text{LOCATEREGION}(\text{PROJ}(s_{goal}))$. Putting it all together, the discrete model is a tuple

$$\mathcal{D} = (\mathcal{W}, \{\mathcal{R}_1, \ldots, \mathcal{R}_n\}, E, \text{LOCATEREGION}, R_{init}, R_{goal}).$$

A solution with respect to the discrete model, referred to as a

high-level plan, is a sequence of regions $[R_{i_j}]_{j=1}^{k}$ connecting R_{init} to R_{goal}.

B. Interplay of High-Level Planning and Sampling-based Motion Planning

Consider a high-level plan $[\mathcal{R}_{i_j}]_{j=1}^{k}$. Sampling-based motion planning in SyCLoP uses $[\mathcal{R}_{i_j}]_{j=1}^{k}$ as a guide in determining the regions that should be further explored. Since $[\mathcal{R}_{i_j}]_{j=1}^{k}$ connects $\mathcal{R}_{init}$ to $\mathcal{R}_{goal}$, by exploring regions in $[\mathcal{R}_{i_j}]_{j=1}^{k}$, the rationale is that sampling-based motion planning can make significant progress in extending $\mathcal{T}$ toward the goal region.

A central issue is then which high-level plan to choose at each iteration (Algo 2:5), since $\mathcal{D}$ can provide exponentially many alternative high-level plans. To address this issue, for each $(\mathcal{R}_i, \mathcal{R}_j) \in \mathcal{D}.E$, SyCLoP maintains a running estimate

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

on the feasibility of having the sampling-based motion planner spend additional time attempting to extend $\mathcal{T}$ 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 feasible spending more time exploring $(\mathcal{R}_i, \mathcal{R}_j)$.

$\textsc{Cost}(\mathcal{R}_i, \mathcal{R}_j)$ is computed based on information gathered by the sampling-based motion planner during each exploration of $\mathcal{R}_i$ and $\mathcal{R}_j$. The topic of estimate computations in sampling-based motion planning has gained considerable attention in recent years [18], [21]–[27], [33], [41]. Drawing from this body of research, earlier versions of our work [23], [24], and extensive experimentations, the 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. In particular, $\textsc{Cost}(\mathcal{R}_i, \mathcal{R}_j)$ depends on the number of times $\mathcal{R}_i$ and $\mathcal{R}_j$ have been explored in the past, the coverage of $\mathcal{R}_i$ and $\mathcal{R}_j$ by $\mathcal{T}$, the free volume of $\mathcal{R}_i$ and $\mathcal{R}_j$, and the progress made in extending $\mathcal{T}$ from $\mathcal{R}_i$ to $\mathcal{R}_j$. Based on this information, $\textsc{Cost}(\mathcal{R}_i, \mathcal{R}_j)$ is estimated to be low, i.e., high feasibility, when the free volume of $\mathcal{R}_i$ and $\mathcal{R}_j$ is high and when the sampling-based motion planner has spent little time exploring $\mathcal{R}_i$ and $\mathcal{R}_j$, but still has made considerable progress in covering and connecting $\mathcal{R}_i$ and $\mathcal{R}_j$. Details are provided in Section III-D.

The computation of a high-level plan $[\mathcal{R}_{i_j}]_{j=1}^{k}$ then essentially becomes a search algorithm on a weighted graph, where $\textsc{Cost}([\mathcal{R}_{i_j}]_{j=1}^{k}) = \textsc{Cost}(\mathcal{R}_{i_1}, \mathcal{R}_{i_2}) + \cdots + \textsc{Cost}(\mathcal{R}_{i_{k-1}}, \mathcal{R}_{i_k})$. Drawing from research in logic and AI [44]–[46], the combination of search strategies in high-level planning (Algo. 2:5) aims to balance greedy and methodical search by selecting more frequently high-level plans with low $\textsc{Cost}([\mathcal{R}_{i_j}]_{j=1}^{k})$, i.e., high feasibility, and selecting less frequently high-level plans with high $\textsc{Cost}([\mathcal{R}_{i_j}]_{j=1}^{k})$. Details are provided in Section III-C.

The core part of SyCLoP, illustrated in Fig. 1 and Algo. 2:4–9 , proceeds by repeating the following steps until a solution is found or a maximum amount of time has elapsed:

- Use high-level planning to compute the current high-level plan $\left[\mathcal{R}_{i_j}\right]_{j=1}^{k}$ by searching $\mathcal{D}$. Use the $\textsc{Cost}(\mathcal{R}_i, \mathcal{R}_j)$ estimates to bias search toward high-level plans with low $\textsc{Cost}([\mathcal{R}_{i_j}]_{j=1}^{k})$, i.e., high feasibility (Algo. 2:5, Section III-C).
- Use sampling-based motion planning to extend $\mathcal{T}$. Use $\left[\mathcal{R}_{i_j}\right]_{j=1}^{k}$ as a guide in determining the regions that should be further explored (Algo. 2:6, Section III-E).
- Update $\textsc{Cost}(\mathcal{R}_i, \mathcal{R}_j)$ estimates based on information gathered by the sampling-based motion planner during exploration (Algo. 2:9, Section III-D).
- Use the updated $\textsc{Cost}(\mathcal{R}_i, \mathcal{R}_j)$ values to discover in future iterations new high-level plans that effectively guide the exploration toward the goal (Algo. 2:5–9).

As demonstrated by the experiments, this synergistic combination of high-level discrete planning and sampling-based motion planning through $\textsc{Cost}(\mathcal{R}_i, \mathcal{R}_j)$ estimates is a crucial component of SyCLoP.

C. High-Level Planning: Guiding the Search

HighLevelPlanning($\mathcal{D}$) (Algo. 2:5) computes at each iteration the current high-level plan by searching the decomposition graph of the discrete model $\mathcal{D}$ for a sequence of regions $[\mathcal{R}_{i_j}]_{j=1}^{k}$ connecting $\mathcal{R}_{i_1} = \mathcal{R}_{init}$ to $\mathcal{R}_{i_k} = \mathcal{R}_{goal}$.

With high probability p (set to 0.95 in the experiments), HighLevelPlanning($\mathcal{D}$) computes the current high-level plan by using a shortest-path algorithm (e.g., Dijkstra, A*), where the edge weights are set to $\textsc{Cost}(\mathcal{R}_i, \mathcal{R}_j)$. This allows SyCLoP to bias the selection toward high-level plans with low $\textsc{Cost}([\mathcal{R}_{i_j}]_{j=1}^{k})$, i.e., high feasibility, which indicate that the sampling-based motion planner should spend 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.

Random high-level plans are also used, although less frequently, as a way to correct for errors inherent with the estimates. This is motivated by observations made in [53], [55], where random restarts and random neighbors have been suggested as effective ways to unblock the exploration when sampling-based motion planners get stuck. With small probability $1 - p$, HighLevelPlanning($\mathcal{D}$) computes the current high-level plan as a random sequence of regions connecting $\mathcal{R}_{init}$ to $\mathcal{R}_{goal}$. The computation is carried out by using depth-first search, where the children are visited in a random order.

D. Definition and Computation of Estimates used by the High-Level Planner and Sampling-based Motion Planner

As discussed in Section III-B, $\textsc{Cost}(\mathcal{R}_i, \mathcal{R}_j)$ estimates in SyCLoP are based on information gathered by the sampling-based motion planner during exploration. These estimates can be defined and computed in a number of ways, as evidenced by recent work in sampling-based motion planning [7], [8], [18], [21]–[27], [33], [41]. Drawing from this research and extensive experimentations, we have made further improvements and have fine-tuned the estimates presented in the preliminary work [23], [24]. As a result, 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. 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 $k \in \{i, j\}$,

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

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.

In this way, $(\mathcal{R}_i, \mathcal{R}_j)$ is more likely to be included in the current high-level plan when $\textsc{Cost}(\mathcal{R}_i, \mathcal{R}_j)$ is low, which indicates that $\mathcal{R}_i$ and $\mathcal{R}_j$ have a large free-volume, the sampling-based motion planner has spent little time exploring $\mathcal{R}_i$ and $\mathcal{R}_j$, but still has been able to make considerable progress in covering and connecting $\mathcal{R}_i$ to $\mathcal{R}_j$. As evidenced by the experimental results, this estimation scheme is particularly well-suited for the interplay between high-level planning and sampling-based motion planning, which has allowed `SyCLoP` to efficiently solve challenging motion-planning problems with dynamics. Details related to the definition and efficient implementation of these estimates follow.

1) Coverage: $\textsc{Cov}(\mathcal{R}_i)$ estimates the coverage of $\mathcal{R}_i$ 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, cf. [56], [57]. One such measure is dispersion. As noted in [58], while dispersion has been used in sampling-based motion planning to generate quasirandom samples [59], its use as a coverage estimate is impeded by the significant cost required to compute it in high dimensions.

As an alternative to dispersion, in order to efficiently compute $\textsc{Cov}(\mathcal{R}_i)$, `SyCLoP` overlays an implicit grid (denoted as *CovGrid*) over the workspace and counts the number of grid cells in $\mathcal{R}_i$ that contain states of vertices v from $\mathcal{T}$, i.e.,

$$\textsc{Cov}(\mathcal{R}_i) = |\{c : c \in \textit{CovGrid} \wedge v \in \mathcal{T} \wedge \textsc{Proj}(v.s) \in c \cap \mathcal{R}_i\}|.$$

The grid is set to a fine resolution to allow for good estimates. Experiments in this paper use a 512×512 (resp., $512 \times 512 \times 512$) grid in 2D (resp., 3D) workspaces.

Note that $\textsc{Cov}(\mathcal{R}_i)$ needs to be updated only when a new vertex v is added to $\mathcal{T}$. To make this update efficient, each region $\mathcal{R}_i$ maintains its own list of coverage cells, $\mathcal{R}_i.\textit{cells}$, and each cell $c \in \mathcal{R}_i.\textit{cells}$ maintains its own list of vertices, $c.\textit{vertices}$. When v is added to $\mathcal{T}$, the lists $\mathcal{R}_i.\textit{cells}$ and $c.\textit{vertices}$ are updated to reflect the new information, as shown in Algo. 3. More specifically, the projection of $v.s$ onto the workspace is computed by $\textsc{Proj}(v.s)$, which is then used to locate the region $\mathcal{R}_i$ in the workspace decomposition where $\textsc{Proj}(v.s) \in \mathcal{R}_i$ (Algo. 3:1). The cell c that $\textsc{Proj}(v.s)$ belongs to is then added to the list $\mathcal{R}_i.\textit{cells}$, if not already there (Algo. 3:2–6). A hash-map is used to efficiently search if c is already in $\mathcal{R}_i.\textit{cells}$ (Algo. 3:3). The vertex v is then associated with c (Algo. 3:7). In this way, $\textsc{Cov}(\mathcal{R}_i)$ can be efficiently updated as

$$\textsc{Cov}(\mathcal{R}_i) = \mathcal{R}_i.\textit{cells}.\textsc{Size}().$$

Algorithm 3 $\textsc{UpdateCoverage}(v)$

1: $\mathcal{R}_i \leftarrow \textsc{LocateRegion}(\textsc{Proj}(v.s))$
2: *coords* $\leftarrow \textsc{CellCoords}(\textit{CovGrid}, \textsc{Proj}(v.s))$
3: $c \leftarrow \mathcal{R}_i.\textit{cells}.\textsc{Get}(\textit{coords})$; *nrNewCells* $\leftarrow 0$
4: **if** $c =$ `null` **then**
5: $\quad c \leftarrow \textsc{CreateCell}(\textit{coords})$; *nrNewCells* $\leftarrow 1$
6: $\quad \mathcal{R}_i.\textit{cells}.\textsc{Add}(c)$
7: $c.\textit{vertices}.\textsc{Add}(v)$
8: $\textsc{Cov}(\mathcal{R}_i) \leftarrow \mathcal{R}_i.\textit{cells}.\textsc{Size}()$
9: **return** $[\mathcal{R}_i, c, \textit{nrNewCells}]$

2) Free Volume: The purpose of the free volume is to provide `SyCLoP` with an estimate on the difficulty of exploring a particular region. The rationale is that regions that have large free volumes are easier to explore than regions that have small free volumes. In a preprocessing stage, as shown in Algo. 4, `SyCLoP` generates a number of samples uniformly at random from $\mathcal{S}$. For each $\mathcal{R}_i \in \{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$, `SyCLoP` then computes the number of valid ($n_{\textit{valid}}(\mathcal{R}_i)$) and invalid ($n_{\textit{invalid}}(\mathcal{R}_i)$) samples that fall into $\mathcal{R}_i$, respectively. (Experiments in this paper use 5000 samples. Preprocessing time is small, less than 3s on a single CPU in our setup.) Then,

Algorithm 4 Preprocessing: $\textsc{FreeVol}$

1: **for** $\mathcal{R}_i \in \{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$ **do**
2: $\quad n_{\textit{valid}}(\mathcal{R}_i) \leftarrow 0$; $n_{\textit{invalid}}(\mathcal{R}_i) \leftarrow 0$
3: **for** $i = 1, \ldots, k$ **do**
4: $\quad s \leftarrow$ generate sample from $\mathcal{S}$
5: $\quad \mathcal{R}_i \leftarrow \textsc{LocateRegion}(\textsc{Proj}(s))$
6: $\quad$ **if** $\textsc{Valid}(s) =$ `true` **then**
7: $\quad\quad n_{\textit{valid}}(\mathcal{R}_i) \leftarrow n_{\textit{valid}}(\mathcal{R}_i) + 1$
8: $\quad$ **else**
9: $\quad\quad n_{\textit{invalid}}(\mathcal{R}_i) \leftarrow n_{\textit{invalid}}(\mathcal{R}_i) + 1$

$$\textsc{FreeVol}(\mathcal{R}_i) = \frac{\epsilon + n_{\textit{valid}}(\mathcal{R}_i)}{\epsilon + n_{\textit{valid}}(\mathcal{R}_i) + n_{\textit{invalid}}(\mathcal{R}_i)}\, \textit{vol}(\mathcal{R}_i),$$

where $\textit{vol}(\mathcal{R}_i)$ is the volume of $\mathcal{R}_i$ and $\epsilon > 0$ is a small constant, which is used to avoid divisions by zero.

3) Connections: $\textsc{Conn}(\mathcal{R}_i, \mathcal{R}_j)$ 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 $\textsc{Proj}(v.s) \in \mathcal{R}_i$ and $\textsc{Proj}(v_{\textit{new}}) \in \mathcal{R}_j$. Then, $\textsc{Conn}(\mathcal{R}_i, \mathcal{R}_j)$ 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 III-D1. Pseudocode is given in Algo. 5. To make this update efficient, 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, $\textsc{Conn}(\mathcal{R}_i, \mathcal{R}_j)$ can be efficiently updated as

$$\textsc{Conn}(\mathcal{R}_i, \mathcal{R}_j) = (\mathcal{R}_i, \mathcal{R}_j).\textit{cells}.\textsc{Size}().$$

Algorithm 5 $\textsc{UpdateConnections}(v, v_{\textit{new}})$

1: $\mathcal{R}_i \leftarrow \textsc{LocateRegion}(\textsc{Proj}(v.s))$
2: $\mathcal{R}_j \leftarrow \textsc{LocateRegion}(\textsc{Proj}(v_{\textit{new}}.s))$
3: **if** $(\mathcal{R}_i, \mathcal{R}_j) \in \mathcal{D}.E$ **then**
4: $\quad \textit{coords} \leftarrow \textsc{CellCoords}(\textit{CovGrid}, \textsc{Proj}(v_{\textit{new}}.s))$
5: $\quad (\mathcal{R}_i, \mathcal{R}_j).\textit{cells} \leftarrow (\mathcal{R}_i, \mathcal{R}_j).\textit{cells} \cup \{\textit{coords}\}$
6: $\textsc{Conn}(\mathcal{R}_i, \mathcal{R}_j) \leftarrow (\mathcal{R}_i, \mathcal{R}_j).\textit{cells}.\textsc{Size}()$

4) Selections: $\textsc{Sel}(\mathcal{R}_i, \mathcal{R}_j)$ distinguishes between "empty" and "nonempty" edges. An edge $(\mathcal{R}_i, \mathcal{R}_j)$ is considered empty if $\mathcal{R}_i$, $\mathcal{R}_j$ have not yet been reached by $\mathcal{T}$, i.e., $\textsc{Cov}(\mathcal{R}_i) = \textsc{Cov}(\mathcal{R}_j) = 0$. In such cases, $\textsc{Sel}(\mathcal{R}_i, \mathcal{R}_j)$ counts the number of times the high-level planner has included $(\mathcal{R}_i, \mathcal{R}_j)$ in the selected high-level plans. This allows the high-level planner to change the empty edges that are included in future high-level plans, giving the sampling-based motion planner a great degree of flexibility during exploration. This is especially relevant in the early stages of exploration, where most of the 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 $\mathcal{T}$, i.e., $\textsc{Cov}(\mathcal{R}_i) \neq 0$ or $\textsc{Cov}(\mathcal{R}_j) \neq 0$, then $\textsc{Sel}(\mathcal{R}_i, \mathcal{R}_j)$ 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 edges that have been explored less frequently in the past.

E. Sampling-based Motion Planning: Guided Exploration

$\textsc{GuidedExploration}(\mathcal{P}, \mathcal{T}, \mathcal{D}, [\mathcal{R}_{i_j}]_{j=1}^k)$ uses the current high-level plan $\left[\mathcal{R}_{i_j}\right]_{j=1}^k$ as a guide in determining the regions that should be further explored, as shown in Algo. 6. At each iteration, a region $\mathcal{R}_i$ is first selected and then explored by extending several branches from the tree vertices associated with $\mathcal{R}_i$ (Algo. 6:3–4). As a result of this exploration, the sampling-based motion planner has gathered new information, which is used to update the estimates, as described in Section III-D and III-E3. These updates allow the sampling-based motion planner to select other regions for exploration during the remaining iterations. If during exploration of $\mathcal{R}_i$, a solution trajectory is found, then $\textsc{GuidedExploration}$ terminates successfully (Algo. 6:5). If exploration of $\mathcal{R}_{i_j}$ improves the overall coverage, i.e. *newCells* > 0, then $\textsc{GuidedExploration}$ continues with the next iteration. This indicates that the sampling-based motion planner is making progress. Otherwise, with small probability p (set to 0.25), $\textsc{GuidedExploration}$ stops exploring the regions associated with the current high-level plan (Algo. 6:6). This indicates that a new high-level plan should be computed, since $\textsc{GuidedExploration}$ is not making additional progress. Details of the main steps in Algo. 6 follow.

Algorithm 6 $\textsc{GuidedExploration}(\mathcal{P}, \mathcal{T}, \mathcal{D}, [\mathcal{R}_{i_j}]_{j=1}^k)$

Input: $\mathcal{P} = (\mathcal{S}, \mathcal{U}, \textsc{Flow}, \textsc{Valid}, s_{init}, \textsc{Goal})$
$\mathcal{T}$: search tree; $\mathcal{D}$: discrete model
$[\mathcal{R}_{i_j}]_{j=1}^k$: current high-level plan
Output: A solution trajectory or `null`

1: $\mathcal{R}_{avail} \leftarrow \textsc{AvailableRegions}([\mathcal{R}_{i_j}]_{j=1}^k)$ ◇ III-E1
2: **for** several times **do**
3: $\quad \mathcal{R}_i \leftarrow \mathcal{R}_{avail}.\textsc{Select}()$ ◇ III-E2
4: $\quad [\gamma, \textit{newCells}] \leftarrow \textsc{ExploreRegion}(\mathcal{P}, \mathcal{T}, \mathcal{D}, \mathcal{R}_i, \mathcal{R}_{avail})$ ◇III-E3
5: $\quad$ **if** $\gamma \neq$ `null` **then return** γ
6: $\quad$ **if** *newCells* $= 0$ **and** $\textsc{rand}[0,1) \leq p$ **then return** `null`
7: **return** `null`

1) $\textsc{AvailableRegions}([\mathcal{R}_{i_j}]_{j=1}^k)$: Note that regions in $\left[\mathcal{R}_{i_j}\right]_{j=1}^k$ which have not yet been reached by $\mathcal{T}$, i.e., $\textsc{Cov}(\mathcal{R}_{i_j}) = 0$, cannot be considered for further exploration, since such regions do not contain any tree vertices from which to extend $\mathcal{T}$. For this reason, the sampling-based motion planner maintains a set of nonempty regions, $\mathcal{R}_{avail}$, which can be considered for further exploration. Initially, $\mathcal{R}_{avail} = \emptyset$. Then, $\left[\mathcal{R}_{i_j}\right]_{j=1}^k$ 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 $\left[\mathcal{R}_{i_j}\right]_{j=1}^k$, since such regions are closer to the goal.

2) $\mathcal{R}_{avail}.\textsc{Select}()$: A region $\mathcal{R}_i$ is selected from $\mathcal{R}_{avail}$ with probability

$$\frac{\mathcal{R}_i.w}{\sum_{\mathcal{R}_j \in \mathcal{R}_{avail}} \mathcal{R}_j.w},$$

where

$$\mathcal{R}_i.w = \frac{\textsc{FreeVol}^4(\mathcal{R}_i)}{(1 + \textsc{Cov}(\mathcal{R}_i))(1 + (\mathcal{R}_i.\textit{nsel})^2)},$$

(for more details on the estimates see discussions and definitions in Section III-D). This selection scheme gives priority to those regions that have a 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.

3) $\textsc{ExploreRegion}(\mathcal{P}, \mathcal{T}, \mathcal{D}, \mathcal{R}_i, \mathcal{R}_{avail})$: Region $\mathcal{R}_i$ is explored by extending several branches from the tree vertices associated with $\mathcal{R}_i$, as shown in Algo. 7. At each iteration, $\textsc{ExploreRegion}$ first selects one of the coverage cells of $\mathcal{R}_i$

Algorithm 7 $\textsc{ExploreRegion}(\mathcal{P}, \mathcal{T}, \mathcal{D}, \mathcal{R}_i, \mathcal{R}_{avail})$

Input: $\mathcal{P} = (\mathcal{S}, \mathcal{U}, \textsc{Flow}, \textsc{Valid}, s_{init}, \textsc{Goal})$
$\mathcal{T}$: search tree; $\mathcal{D}$: discrete model
$\mathcal{R}_i$: region to be explored; $\mathcal{R}_{avail}$: available regions
Output: A solution trajectory or `null`

1: **for** several times **do**
2: $\quad c \leftarrow \mathcal{R}_i.\textit{cells}.\textsc{Select}()$
3: $\quad v \leftarrow c.\textit{vertices}.\textsc{Select}()$
4: $\quad \textsc{UpdateOnSelect}(\mathcal{R}_i, c, v)$
5: $\quad \gamma \leftarrow \textsc{ExtendTree}(\mathcal{P}, \mathcal{T}, v)$; **if** $\gamma \neq$ `null` **then return** $[\gamma, 0]$
6: $\quad \textit{newCells} \leftarrow 0$
7: $\quad$ **for** v_{new} added by $\textsc{ExtendTree}$ **do**
8: $\quad\quad \textit{newCells} \leftarrow \textsc{UpdateOnNewVertex}(v_{new}) + \textit{newCells}$
9: $\quad$ **if** *newCells* $= 0$ **and** $\textsc{rand}(0,1) \leq p$ **then**
10: $\quad\quad$ **return** [`null`, *newCells*]
11: **return** [`null`, *newCells*]

$\textsc{UpdateOnNewVertex}(v_{new})$
1: $[\mathcal{R}_i, \textit{newCells}] \leftarrow \textsc{UpdateCoverage}(v_{new})$
2: $\textsc{UpdateConnections}(v_{new}.\textit{parent}, v_{new})$
3: **if** $\mathcal{R}_{avail}.\textsc{Exists}(\mathcal{R}_i) =$ `false` **then**
4: $\quad \mathcal{R}_{avail}.\textsc{Add}(\mathcal{R}_i)$
5: $\mathcal{R}_{avail}.\textsc{Update}(\mathcal{R}_i)$
6: **return** *newCells*

$\textsc{UpdateOnSelect}(\mathcal{R}_i, c, v)$
1: $v.\textit{nsel} \leftarrow v.\textit{nsel} + 1$; $c.\textit{vertices}.\textsc{Update}(v)$
2: $c.\textit{nsel} \leftarrow c.\textit{nsel} + 1$; $\mathcal{R}_i.\textit{cells}.\textsc{Update}(c)$
3: $\mathcal{R}_i.\textit{nsel} \leftarrow \mathcal{R}_i.\textit{nsel} + 1$; $\mathcal{R}_{avail}.\textsc{Update}(\mathcal{R}_i)$

(Section III-D1) with probability

$$\frac{1}{1+c.nsel} \Big/ \sum_{c' \in \mathcal{R}_i.cells} \frac{1}{1+c'.nsel},$$

where $c.nsel$ is the number of times the coverage cell c has been selected in the past (Algo. 7:2). Then, a vertex is selected from $c.vertices$ using a similar probability distribution, i.e.,

$$\frac{1}{1+v.nsel} \Big/ \sum_{v' \in c.vertices} \frac{1}{1+v'.nsel},$$

where $v.nsel$ denotes the number of times v has been selected in the past (Algo. 7:3). Estimates are then updated to reflect these new selections (Algo. 7:4). In this way, a vertex v that has been selected less frequently during past explorations of $\mathcal{R}_i$ has a higher likelihood of being selected during the current exploration of $\mathcal{R}_i$. This two-tier selection process has been proposed in [41] and has been shown to work well in practice.

EXTENDTREE$(\mathcal{P}, \mathcal{T}, v)$ is then used to extend $\mathcal{T}$ from v (Algo. 7:5) by forward propagation of the system dynamics (see Section II-B). As a result, new vertices might have been added to $\mathcal{T}$. If a new region, $\mathcal{R}_{new}$, was reached, it is then added to $\mathcal{R}_{avail}$, so that it becomes available for further exploration (Algo. 7:7-8). Moreover, the estimates are also updated to reflect the new information gathered by the sampling-based motion planner (Algo. 7:7–8).

If EXTENDTREE$(\mathcal{P}, \mathcal{T}, v)$ improves the overall coverage, i.e., $newCells > 0$, then the exploration of $\mathcal{R}_i$ continues. Otherwise, with small probability p (set to 0.125), the exploration of $\mathcal{R}_i$ stops (Algo. 7:9–10). In this way, the sampling-based motion planner spends more time exploring regions that improve the overall coverage and spends less time exploring regions that have already been covered well.

4) Implementation of the Select Functions: As discussed, $\mathcal{R}_{avail}$.SELECT(), $cells$.SELECT(), $vertices$.SELECT() operate by selecting an item a_i from a collection of items $\{a_1, \ldots, a_n\}$ with probability $a_i.w / \sum_{j=1}^{n} a_j.w$, where $a_i.w$ is a positive weight associated with a_i. A straightforward implementation of SELECT() can be obtained by selecting a weight w uniformly at random from $[0, w_{total}]$, where $w_{total} = a_1.w + \cdots + a_n.w$, and then iterating from $i = 1$ to n until $w \geq a_1.w + \cdots + a_i.w$. The total weight, w_{total}, is maintained current after each item addition or weight update. This straightforward implementation, however, works well only for small collections. More efficient implementations of SELECT() with $O(\log(n))$ time, which are used in this paper, can be obtained by placing $a_1, \ldots, a_n$ as leaves in a weighted complete binary tree, where the weight of each inner node (start construction from bottom to top) is equal to the sum of weights of its children. In this case, item additions and weight updates also take $O(\log(n))$ time.

IV. EXPERIMENTS AND RESULTS

This section demonstrates the computational efficiency of SyCLoP in solving challenging high-dimensional motion-planning problems with dynamics. The experiments also indicate that the synergistic combination of high-level discrete planning and sampling-based motion planning is a crucial component in the computational efficiency of SyCLoP. This section also studies the role of the workspace decomposition in this synergistic combination.

A. Experimental Setup

1) Sampling-based Motion Planners used in the Comparisons: SyCLoP is compared to state-of-the-art and widely popular methods, such as RRT [5], [6], ADDRRT [16], EST [7], [8], and SBL [41]. We note that in all the experiments, SBL, which is a more recent version of EST, outperformed the original EST. Taking this into account, the results in this section only include comparisons to RRT, ADDRRT, and SBL.

Standard implementations were followed, as suggested in the respective research papers and motion-planning books [10], [11]. These implementations are based on the tree-search framework (Algo. 1) and are referred to as RRT[TSF], ADDRRT[TSF], and SBL[TSF]. Many of the data structures and utilities available in OOPSMP [60] were used to facilitate implementation. Every effort was made to fine-tune the performance of these motion planners for problems with dynamics.

RRT[TSF]: This implementation uses goal bias (set to 0.05), which has been shown to improve the efficiency of RRT [5], [6]. The distance metric was set to Euclidean distance on the position component of the state. Other metrics, e.g., weighted combination of distances on position, orientation, and velocity components, did not work as well.

ADDRRT[TSF]: This extends RRT[TSF] by implementing SELECTVERTEXFROMTREE$(\mathcal{P}, \mathcal{T})$ as in [16].

SBL[TSF]: SBL [41] was developed for geometric path planning. As a result, not all components of SBL can be extended and used in the context of motion planning with dynamics. The SBL implementation in this work is obtained by using the tree-search framework (Algo. 1) and implementing SELECTVERTEXFROMTREE$(\mathcal{P}, \mathcal{T})$ as described in [41].

2) Models of Ground and Flying Vehicles with Second-Order Dynamics: This paper contains experiments with second-order dynamical models of cars, planar body thrusters, unicycles, "flying" unicycle, and high-dimensional tractor-trailers. The 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.

Car (adapted from [11, 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.

Planar Body with Two Thrusters (adapted from [10, 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 $2L$.

Unicycle (adapted from [11, 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 ($|\omega| \leq 100^\circ/s$). The unicycle is controlled by setting the translational u_0 ($|u_0| \leq 1m/s^2$) and rotational u_1 ($|u_1| \leq 25^\circ/s^2$) accelerations. The 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 $0.25m$, respectively.

"Flying" Unicycle (adapted from [18]): 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.

Tractor-Trailer (adapted from [11, 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 [30].

3) Measuring the Computational Efficiency: For each motion-planning benchmark, the computational efficiency of a motion planner, denoted as $t_{MotionPlanner}$, is measured as the median time to solve 30 random queries.

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

B. Computational Efficiency of SyCLoP

1) Experiments on Motion-Planning Problems with Dynamical Models of Ground Vehicles: Fig 2 summarizes the results of the experiments with several motion-planning benchmarks and second-order models of ground vehicles, such as unicycles, thrusters, and cars (Section IV-A2). The benchmarks are chosen to vary in difficulty, where 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 obtains significant computational speedups over RRT[TSF], ADDRRT[TSF], and SBL[TSF]. As an example, in the case of the thruster benchmark (Fig 2(b)), SyCLoP is 37, 31, and 7 times faster than RRT[TSF], ADDRRT[TSF], and SBL[TSF], respectively. Moreover, the efficiency of SyCLoP becomes more pronounced as harder problems are considered, as indicated by the rest of the experiments.

2) Experiments 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 described in Section IV-A2. Preliminary work [23], [24] did not contain such experiments.

The three-dimensional workspace consists of several walls placed consecutively at a distance from each-other, all parallel to the XZ-plane. In each wall, nonoverlapping small holes are placed at random positions. The opening of each hole along the X and Z dimensions 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 [53], provides several options of varying difficulty to pass from one side of the wall to the other. Fig. 3 provides an illustration of one wall 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 is significantly faster than RRT[TSF], ADDRRT[TSF], and SBL[TSF]. Moreover, the computational efficiency of SyCLoP becomes more pronounced as the number of walls is increased. In fact, RRT[TSF] and ADDRRT[TSF] time out ($t_{\texttt{RRT[TSF]}}, t_{\texttt{ADDRRT[TSF]}} \geq 700$s) at instances with 6 walls. Even though SBL[TSF] does not time out, it still has a high computational cost: $t_{\texttt{SBL[TSF]}} = 670.20$s. In contrast, SyCLoP efficiently solves such problems, i.e., $t_{\texttt{SyCLoP}} = 61.90$s.

3) Experiments on Motion-Planning Problems with High-Dimensional Dynamical Models: The robot consists of a second-order dynamical model of a tractor-trailer (Section IV-A2). By adding more trailers to increase DOFs, the tractor-trailer provides a challenging high-dimensional problem [30]. Recall that number of DOFs is $5+N$, where N is the number of trailers. We note that preliminary work [23], [24] did not contain experiments with high-dimensional models.

Fig. 4 contains a summary of the experiments. In these experiments, the number of trailers attached to the tractor varies from 1 to 20, yielding problems with $6, \ldots, 25$ DOFs. As shown in Fig. 4, SyCLoP is significantly faster than RRT[TSF], ADDRRT[TSF], and SBL[TSF]. As an example, on motion-planning problem instances with 15 DOFs (10 trailers attached to the tractor), $t_{\texttt{RRT[TSF]}} = 262.33$s, $t_{\texttt{ADDRRT[TSF]}} = 357.57$s, and $t_{\texttt{SBL[TSF]}} = 482.31$s, while $t_{\texttt{SyCLoP}} = 29.01$s. Moreover, the computational efficiency of RRT[TSF], ADDRRT[TSF], and SBL[TSF] deteriorates rapidly as the number of DOFs is

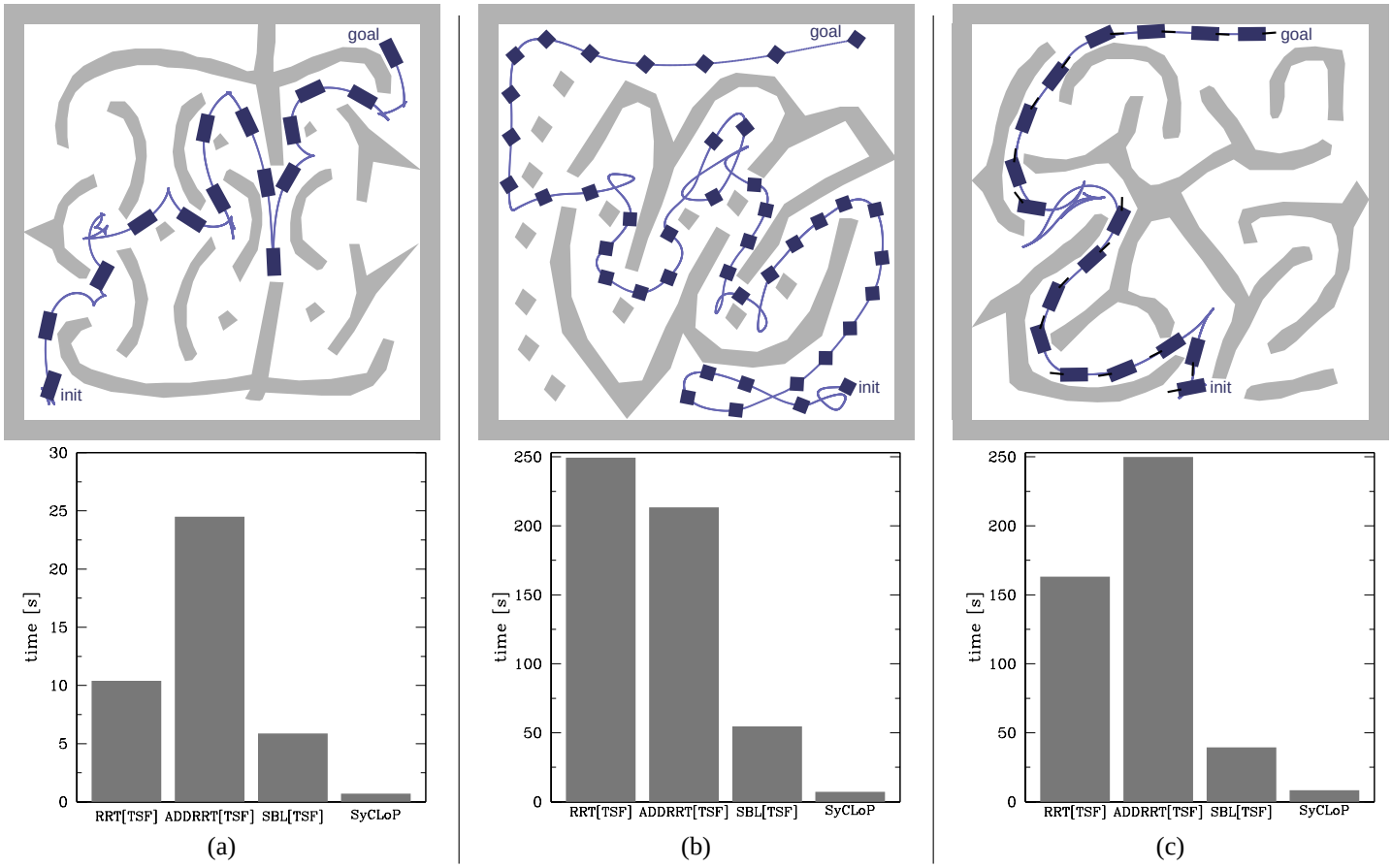

Fig. 2. Results of the experiments with second-order dynamical models of ground vehicles: (a) unicycle, (b) thruster, (c) car. The top portion of each column illustrates the workspace (shown in light gray) and a solution trajectory to a typical query (trajectory shown in dark color as an (x, y) curve 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 motion planner, measured as the median computational time in solving 30 queries. In these experiments, SBL[TSF] and SyCLoP use a 32×32 uniform-grid decomposition of the workspace.

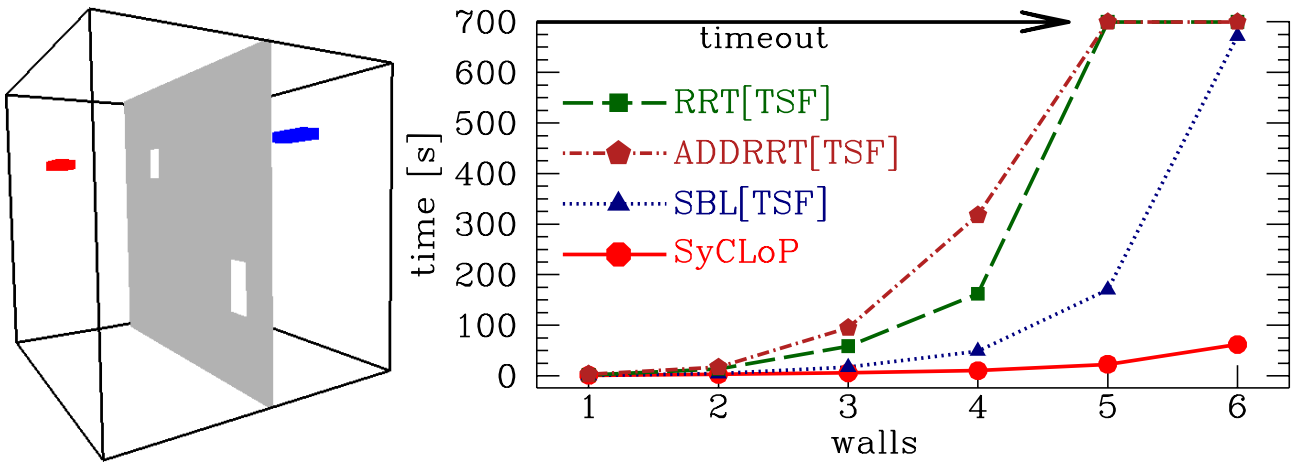

Fig. 3. Results of the experiments with a second-order dynamical model of a flying unicycle. (left) An example of the workspace with one wall (shown in gray) and two small holes. A typical random query (initial and goal placements shown in blue and red) is also shown on the same figure. (right) Results 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 motion planner is measured as the median computational time in solving 30 queries. The maximum running time for each query is set to 700s. In these experiments, SBL[TSF] and SyCLoP use a $32\times32\times32$ uniform-grid decomposition of the three-dimensional workspace.

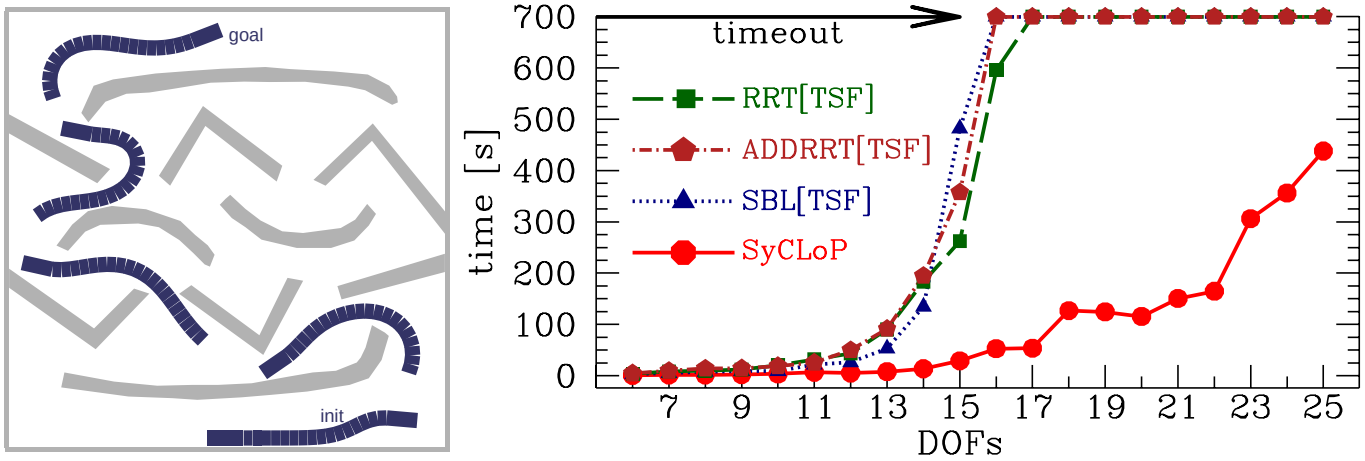

Fig. 4. Results of the experiments with a second-order dynamical model of a tractor-trailer. (left) A typical workspace (obstacles shown in light gray) and a solution trajectory to a typical query (trajectory shown in dark as several intermediate robot configurations. Other state values, e.g., velocity, are not shown.) The illustration corresponds to a tractor pulling 20 trailers, a 25 DOF problem. (right) Computational efficiency of each motion planner as a function of the number of DOFs, measured as the median computational time in solving 30 queries. The maximum running time for each query is set to 700s. In these experiments, `SBL[TSF]` and `SyCLoP` use a 32×32 uniform-grid decomposition of the workspace.

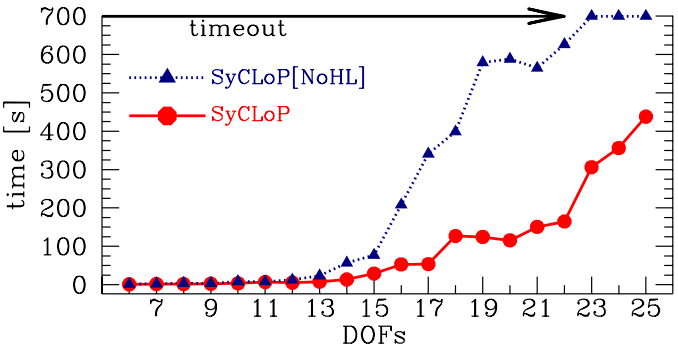

Fig. 5. Impact of the synergistic combination of high-level discrete planning and sampling-based motion planning. `SyCLoP[NoHL]` corresponds to the motion-planning layer of `SyCLoP`, excluding the interactive combination with the high-level discrete planning layer. Results are shown for the motion-planning benchmark with the tractor-trailer dynamical model (see Section IV-B3), where the number of trailers is varied from 1 to 20 (corresponding to 6 to 25 DOFs). Computational efficiency of each motion planner as a function of the number of DOFs, measured as the median computational time in solving 30 queries. The maximum running time for each query is set to 700s. In these experiments, `SyCLoP[NoHL]` and `SyCLoP` use a 32×32 uniform-grid decomposition of the workspace.

increased. In fact, `RRT[TSF]`, `ADDRRT[TSF]`, and `SBL[TSF]` time out (set to 700s) at problems with 17 DOFs, 16 DOFs, and 16 DOFs, respectively. In contrast, `SyCLoP` solves such problems quite fast ($t_{\texttt{SyCLoP}} = 53.63$s) and effectively handles much higher dimensional problems ($t_{\texttt{SyCLoP}} = 438.24$s for the 25 DOFs problem instances).

C. Impact of the Synergistic Combination of High-Level Discrete Planning and Sampling-based Motion Planning

The main strength of `SyCLoP` is the synergistic combination of high-level discrete planning and sampling-based motion planning. To quantify this observation, Fig. 5 compares `SyCLoP` to `SyCLoP[NoHL]`. `SyCLoP[NoHL]` corresponds to sampling-based motion planning in `SyCLoP`, excluding the synergistic combination with the high-level discrete planning. More specifically, `SyCLoP[NoHL]` is obtained by minor modifications to GUIDEDEXPLORATION (Algo. 6), as shown below:

`SyCLoP[NoHL]`
1: $\mathcal{R}_{avail} \leftarrow$ LOCATEREGION(PROJ($v_{init}.s$))
2: **while** ELAPSEDTIME $< t_{max}$ **do**
3: $\quad \mathcal{R}_i \leftarrow \mathcal{R}_{avail}$.SELECT()
4: $\quad [\gamma, newCells] \leftarrow$ EXPLOREREGION($\mathcal{R}_i, \mathcal{R}_{avail}$)
5: $\quad$ **if** $\gamma \neq$ `null` **then return** γ
6: **return** `null`

The results in Fig. 5 indicate that without this synergistic combination, the computational efficiency of the sampling-based motion planner deteriorates quickly. In fact, `SyCLoP[NoHL]` fails to solve many of the high-dimensional problems that `SyCLoP` can solve efficiently. This is to be expected, since `SyCLoP` showed significant computational speedups in comparison to other state-of-the-art sampling-based motion planners, i.e., `RRT[TSF]`, `ADDRRT[TSF]`, and `SBL[TSF]`. As detailed in Section III, high-level plans guide the sampling-based motion planner to extend the tree closer to the goal. The exploration provides valuable feedback information that is used by `SyCLoP` to refine the high-level plan for the next motion-planning step. As the search progresses, the high-level plans become increasingly feasible and guide the motion planner closer to the goal until it eventually reaches the goal. This combination of high-level planning and sampling-based motion planning makes it possible for `SyCLoP` to efficiently solve challenging high-dimensional problems with dynamics.

D. Impact of Workspace Decomposition

As described in Section III-A, the workspace decomposition provides a simplified high-level discrete model, which is used by `SyCLoP` to combine high-level discrete planning with sampling-based motion planning. This section provides a quantitative study of the impact of the workspace decomposition on the computational efficiency of `SyCLoP`. We note that our preliminary work [24], which carried out a similar study, used different test cases, and in particular, it did not contain experiments with high-dimensional models.

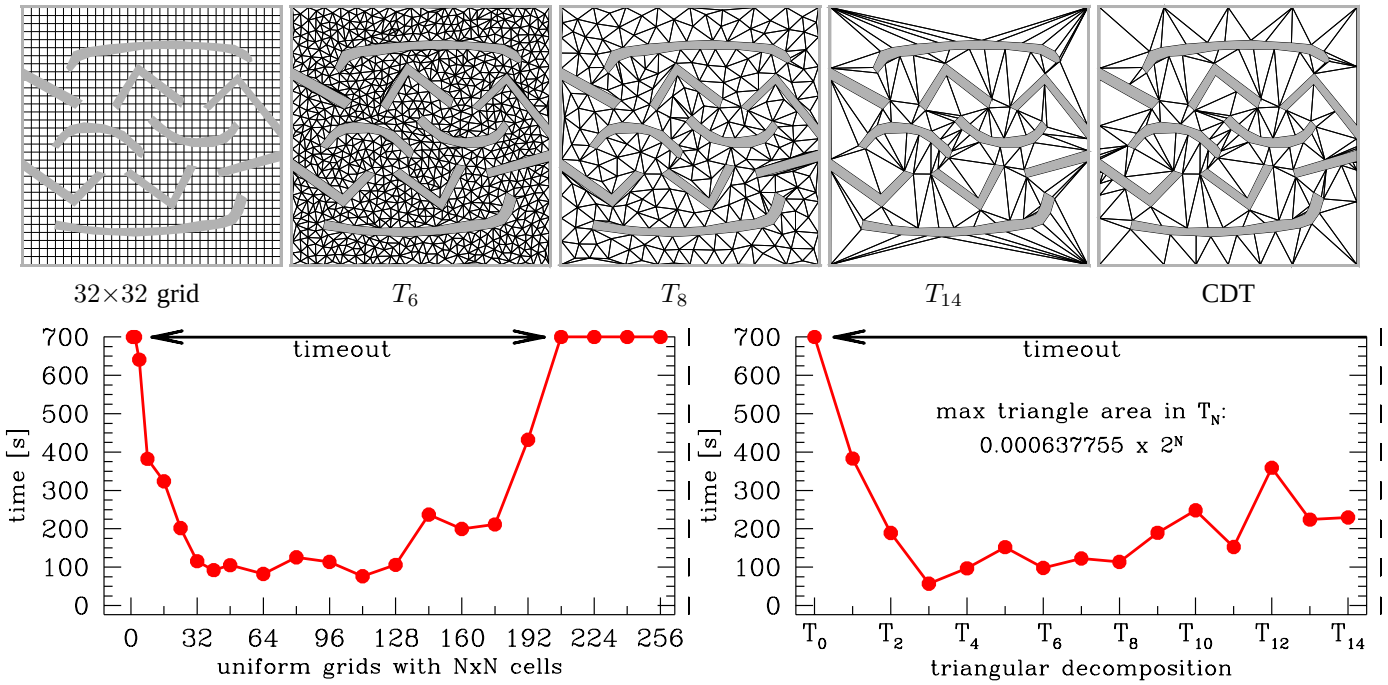

Fig. 6. Computational efficiency of SyCLoP as a function of the decomposition granularity. (top) An illustration of the 32×32 grid decomposition and several triangulations. (bottom) Results are shown for the motion-planning benchmark with the tractor-trailer dynamical model (see Section IV-B3), where 15 trailers are attached to the tractor (corresponding to 20 DOFs).

Grid Decompositions: As noted, the results presented in Section IV-B 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 IV-B 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 IV-B 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 [61] package was used for the computations.

Fig. 6 shows a summary of the results obtained for the motion-planning problem with the second-order dynamical model of a tractor-trailer (Section IV-B3) with 15 trailers attached to the tractor (20 DOFs). Fig. 6 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. 6 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 IV-B (not shown here due to space limitations).

As an alternative to finding the optimal decomposition gran-

TABLE I
COMPUTATIONAL EFFICIENCY OF SyCLoP WHEN USING CONFORMING DELAUNAY TRIANGULATIONS (CDT) COMPARED TO OTHER SAMPLING-BASED MOTION PLANNERS

	t_{SyCLoP} [CDT]	t_{RRT} [TSF]	t_{ADDRRT} [TSF]	t_{SBL} [TSF]
unicycle: Fig. 2(a)	0.73s	10.39s	24.50s	5.87s
thruster: Fig. 2(b)	12.73s	249.24s	213.37s	54.66s
car: Fig. 2(c)	8.68s	163.24s	249.96s	39.46s
tractor-trailer: Fig. 4				
14 DOFs	20.03s	183.51s	194.97s	134.60s
16 DOFs	58.45s	595.70s	X	X
18 DOFs	171.18s	X	X	X
20 DOFs	275.22s	X	X	X
22 DOFs	252.59s	X	X	X
24 DOFs	490.51s	X	X	X

Entries marked with X indicate a timeout, which was set to 700s per query.

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

	$t_{\text{SyCLoP[CDT]}}/$ $t_{\text{SyCLoP[best-tri]}}$	$t_{\text{SyCLoP[CDT]}}/$ $t_{\text{SyCLoP[best-grid]}}$
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 [$G_{48\times48}$]

$t_{\text{SyCLoP[CDT]}}$ denotes the computational efficiency of SyCLoP when using conforming Delaunay triangulation.

$t_{\text{SyCLoP[best-grid]}}$ denotes the best computational efficiency achieved by SyCLoP when using one of the grid decompositions.

$t_{\text{SyCLoP[best-tri]}}$ denotes the best computational efficiency achieved by SyCLoP when using one of the triangular decompositions $T_0, \ldots, T_{14}$. Decomposition that achieves the best computational efficiency is denoted inside parentheses

ularity, 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) [61], [62].

Table I summarizes the results obtained by SyCLoP when using a CDT, denoted by SyCLoP[CDT], in comparison to other sampling-based motion planners. As shown in Table I, SyCLoP[CDT] is significantly faster than RRT[TSF], ADDRRT[TSF], and SBL[TSF]. As an example, on tractor-trailer problems with 14 DOFs, SyCLoP[CDT] is one-order of magnitude faster than RRT[TSF], ADDRRT[TSF], and SBL[TSF]. As the dimensionality of the problem increases, the computational efficiency of SyCLoP[CDT] becomes even more pronounced as RRT[TSF], ADDRRT[TSF], and SBL[TSF] time out at problems with 18, 16, 16 DOFs, respectively. One reason for the efficiency of SyCLoP[CDT] is that a CDT produces triangulations that eliminate narrow angles as much as possible. As a result, it is easier for the sampling-based motion planner to cover the triangle during exploration. Moreover, a CDT provides high-level plans that do not go through workspace obstacles, which further facilitates explorations.

Table II summarizes results obtained by SyCLoP when using CDTs in comparison to SyCLoP when using other triangular or grid decompositions. Table II shows that the computational efficiency of SyCLoP[CDT] is comparable to the best computational efficiency that SyCLoP achieves by using other triangular or grid decompositions. In this way, CDTs provide workspace decompositions that require no fine-tuning and allow SyCLoP to obtain close to optimal computational efficiency.

V. Applications of SyCLoP to Motion Planning for Hybrid Systems

Although not the focus of this paper, SyCLoP is also well-suited for hybrid systems. Hybrid systems go beyond continuous models by employing discrete logic to instantaneously modify the underlying robot dynamics and switch to a different mode in order to respond to mishaps or unanticipated changes. While the combination of discrete logic and continuous dynamics poses significant challenges to current motion-planning methods, it is particularly well-suited to SyCLoP, which synergistically combines high-level discrete planning with sampling-based motion planning. The work in [49] builds upon the idea of combining high-level discrete planning and sampling-based motion planning to effectively solve challenging high-dimensional problems for hybrid systems. Experiments on a nonlinear hybrid robotic system with over one million modes and experiments with an aircraft conflict-resolution protocol with high-dimensional continuous state spaces (60 dimensions) show computational speedups of up to two orders of magnitude over related work.

VI. Discussion

To effectively solve challenging motion-planning problems with dynamics, this paper developed a multi-layered approach, SyCLoP, that synergistically combined high-level discrete planning and sampling-based motion planning. High-level discrete planning guides the sampling-based motion planning during the search for a solution. Information gathered during the search is in turn fed back from the sampling-based motion planner to the high-level planner in order to compute increasingly feasible high-level plans in future iterations. In this way, high-level plans become increasingly useful in guiding the sampling-based motion planner toward a solution. Simulation experiments on high-dimensional motion-planning problems with second-order dynamical models of ground- and flying-robotic vehicles demonstrated significant computational speedup of up to two orders of magnitude over state-of-the-art motion planners.

As we consider increasingly challenging problems, it becomes important to make use of parallel or multi-threaded computational resources in order 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.

Acknowledgment

This work has been supported in part by NSF CNS 0615328 (EP, LK,MV), 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, "A survey of motion planning and related geometric algorithms," *Artificial Intelligence*, vol. 37, pp. 157 – 169, 1988.

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

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

[9] J.-C. Latombe, *Robot Motion Planning*. Boston, MA: Kluwer, 1991.

[10] 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.

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

[12] 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.

[13] J. Laumond, P. Jacobs, M. Taix, and R. Murray, "A motion planner for nonholonomic mobile robots," *IEEE Trans. Robot. Autom.*, vol. 10, pp. 577–593, 1994.

[14] F. Lamiraux and J. Laumond, "Smooth motion planning for car-like vehicles," *IEEE Trans. Robot. Autom.*, vol. 17, no. 4, pp. 498–501, 2001.

[15] E. Frazzoli, M. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," *J. of Guidance Control and Dynamics*, vol. 25, no. 1, pp. 116–129, 2002.

[16] 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.

[17] 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.

[18] 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.

[19] M. Kalisiak and M. van de Panne, "RRT-blossom: RRT with a local flood-fill behavior," in *IEEE Int. Conf. Robot. Autom.*, Orlando, FL, 2006, pp. 1237–1242.

[20] R. Alterovitz, T. Simeon, and K. Goldberg, "The Stochastic Motion Roadmap: A Sampling Framework for Planning with Markov Motion Uncertainty," in *Robot.: Sci. Syst.*, Atlanta, GA, 2007.

[21] 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.

[22] T. Nahhal and T. Dang, "Test coverage for continuous and hybrid systems," in *Int. Conf. Comp. Aided Verif.*, ser. Lecture Notes in Comp. Sci., Berlin, Germany, 2007, vol. 4590, pp. 449–462.

[23] 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.

[24] ——, "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.

[25] I. A. Şucan, J. F. Kruse, M. Yim, and L. E. Kavraki, "Kinodynamic motion planning with hardware demonstration," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, Nice, France, 2008, pp. 1161–1166.

[26] I. A. Şucan and L. E. Kavraki, "Kinodynamic motion planning by interior-exterior cell exploration," in *Int. Work. Algo. Found. Robot.*, Guanajuato, Mexico, 2008.

[27] K. Tsianos and L. E. Kavraki, "Replanning: A powerful planning strategy for hard kinodynamic problems," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, Nice, France, 2008, pp. 1667–1672.

[28] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. John Wiley and Sons, 2005.

[29] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *Journal of the ACM*, vol. 40, pp. 1048–1066, 1993.

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

[31] J. Laumond and J. Risler, "Nonholonomic systems: controllability and complexity," *Theor. Comp. Sci.*, vol. 157, pp. 101–114, 1996.

[32] P. Cheng, G. Pappas, and V. Kumar, "Decidability of motion planning with differential constraints," in *IEEE Int. Conf. Robot. Autom.*, Rome, Italy, 2007, pp. 1826–1831.

[33] 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.

[34] M. Rickert, O. Brock, and A. Knoll, "Balancing exploration and exploitation in motion planning," in *IEEE Int. Conf. Robot. Autom.*, Pasadena, CA, 2008, pp. 2812–2817.

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

[36] D. Zhu and J.-C. Latombe, "New heuristic algorithms for efficient hierarchical path planning," *IEEE Trans. Robot. Autom.*, vol. 7, pp. 9–20, 1991.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

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

[43] Y. Yang and O. Brock, "Efficient motion planning based on disassembly," in *Robotics: Sci. and Systems*, Cambridge, MA, 2005, pp. 97–104.

[44] S. russle and P. Norvig, *Artifical Intelligence: A Modern Approach*, 2nd ed. Prentic Hall, 2002.

[45] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. MIT Press, 2000.

[46] W. Zhang, *State-space Search: Algorithms, Complexity, Extensions, and Applications*. New York, NY: Springer Verlag, 2006.

[47] P. Chen and Y. Hwang, "SANDROS: A motion planner with performance proportional to task difficulty," in *ICRA*, Nice, France, 1992, pp. 2346–2353.

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

[49] 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*, 2008.

[50] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The algorithmic analysis of hybrid systems," *Theor. Comp. Sci.*, vol. 138, no. 1, pp. 3–34, 1995.

[51] C. J. Tomlin, I. Mitchell, A. Bayen, and M. Oishi, "Computational techniques for the verification and control of hybrid systems," *Proceedings of the IEEE*, vol. 91, no. 7, pp. 986–1001, 2003.

[52] A. Bicchi and L. Pallottino, "On optimal cooperative conflict resolution for air-traffic management systems," *IEEE Trans. Intell. Transp. Syst.*, vol. 1, no. 4, pp. 221–231, 2000.

[53] 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.

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

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

[56] H. Niederreiter, *Random Number Generation and Quasi-Monte Carlo Methods*. Society for Industrial and Applied Mathematics, 1992.

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

[58] 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.

[59] 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.

[60] 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.

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

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