Adaptive Sampling-Based Motion Planning for Mobile Robots with Differential Constraints

Andrew Wells and Erion Plaku(✉)

Department of Electrical Engineering and Computer Science,
Catholic University of America, Washington, DC, USA
plaku@cua.edu

Abstract. This paper presents a sampling-based motion planner geared towards mobile robots with differential constraints. The planner conducts the search for a trajectory to the goal region by using sampling to expand a tree of collision-free and dynamically-feasible motions. To guide the tree expansion, a workspace decomposition is used to partition the motion tree into groups. Priority is given to tree expansions from groups that are close to the goal according to the shortest-path distances in the workspace decomposition. To counterbalance the greediness of the shortest-path heuristic, when the planner fails to expand the tree from one region to the next, the costs of the corresponding edges in the workspace decomposition are increased. Such cost increases enable the planner to quickly discover alternative routes to the goal when progress along the current route becomes difficult or impossible. Comparisons to related work show significant speedups.

Keywords: Sampling-based motion planning · Differential constraints · Workspace decomposition · Discrete search

1 Introduction

In motion planning, the objective is to compute a collision-free and dynamically-feasible trajectory which enables the robot to reach a desired goal region while avoiding collisions with obstacles. Motion planning arises in numerous robotics applications ranging from autonomous vehicles, manufacturing, surveillance to robotic-assisted surgery, air-traffic control, and computer animations [1,5,13].

In order to facilitate the execution of the planned motions, the motion planner needs to take into account the underlying robot dynamics. Robot dynamics express physical constraints on the feasible motions, such as minimum turning radius, directional stability, bounded curvature, velocity, and acceleration. Robot dynamics are often modeled as differential equations of the form $\dot{s} = f(s, u)$ to indicate the changes that occur in the state s as a result of applying the input control u, e.g., setting the acceleration and steering angle. Such differential constraints make motion planning challenging as the planner needs to find controls that result in a motion trajectory to the goal that is not only collision-free but also dynamically-feasible so that it can be followed by the robot.

C. Dixon and K. Tuyls (Eds.): TAROS 2015, LNAI 9287, pp. 283–295, 2015.
DOI: 10.1007/978-3-319-22416-9_32

To take into account the constraints imposed by the robot dynamics and obstacles, this paper draws from sampling-based motion planning which has shown promise in solving challenging problems [5,13]. In a sampling-based formulation, motion planning is defined as a search problem. To conduct the search, a motion tree is rooted at the initial state and is incrementally expanded by adding collision-free and dynamically-feasible trajectories as branches. The proposed approach uses a workspace decomposition to obtain a simplified planning layer which serves to guide the sampling-based search. The workspace decomposition is used to induce a partition of the motion tree into groups and to determine routes along which to expand each group. In particular, each group is defined by a region r in the decomposition and the motion-tree vertices that are in r. The search is driven by procedures to (i) select a group, (ii) expand the motion tree from the selected group, and (iii) update heuristic costs to reflect the progress made. To promote expansions toward the goal, priority is given to groups that are close to the goal according to the shortest-path distances in the workspace decomposition. After selecting a group, attempts are made to expand the motion tree along the shortest path in the workspace decomposition to the goal region. To counterbalance the greediness of the shortest-path heuristic, when the group expansion fails, the costs of the corresponding edges in the workspace decomposition are increased. Such cost increases enable the proposed approach to quickly discover alternative routes to the goal when progress along the current route becomes difficult or impossible. Experimental validation is provided using high-dimensional models with nonlinear dynamics where the robot has to operate in complex environments and wiggle its way through narrow passages in order to reach the goal. Comparisons to related work show significant speedups.

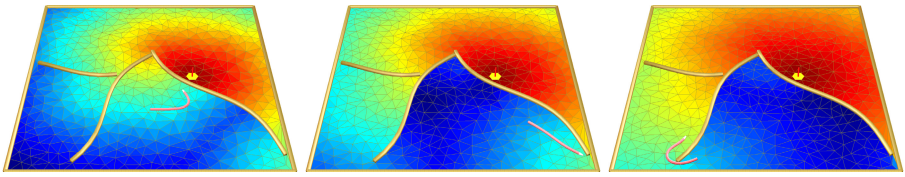
2 Related Work

The discussion of related work focuses on sampling-based motion planning since it is the basis for the proposed approach. Over the years, numerous approaches have been proposed on how to conduct a sampling-based search. RRT [15], which is one of the most widely used planners, and its variants [11,14] expand the motion tree from the nearest vertex to a random sample to bias the search toward the largest uncovered Voronoi regions. Other approaches rely on probability distributions [10], subdivisions [6], principal-component analysis [7], stochastic transitions [8], and density estimations [9].

Even though considerable progress has been made, it remains challenging to effectively incorporate robot dynamics into planning. As the problem dimensionality increases and the robot dynamics become more complex, the search in many of these approaches becomes less and less efficient [5,13].

To improve the computational efficiency, Syclop [17,19] coupled discrete search over a workspace decomposition with sampling-based motion planning. Subsequent work [16,18] offered further improvements by partitioning the motion-tree into groups and using shortest-path distances as heuristic costs to effectively guide the motion-tree expansion. Comparisons to RRT and other

planners showed significant speedups. The approach proposed in this paper builds upon this line of work which couples discrete search with sampling-based motion planning. The proposed approach addresses the shortcomings observed in Syclop [19] and its variants [16,18] when dealing with complex problems. In particular, these planners, due to the greediness of the shortest-path heuristic, spend considerable time before realizing that the current guide should be abandoned due to constraints imposed by obstacles and robot dynamics. Fig. 1 shows an illustration where the shortest-path heuristic leads Syclop and its variants along an infeasible route. Other examples are shown in Fig. 3. The proposed approach addresses such shortcomings by counterbalancing the greediness of the shortest-path heuristic with edge penalties. When the tree expansion fails to make progress, the costs of the corresponding edges in the workspace decomposition are increased. Such cost increases enable the proposed approach to quickly discover alternative routes to the goal. As illustrated in Fig. 1, after failing to move the robot through the impassible openings at the top and on the right, the increases in the edge costs quickly enable the proposed approach to discover a feasible route to the goal by passing through the opening on the left side. Comparisons to prior work show significant speedups.



(a) (b) (c)

Fig. 1. Illustration of how the proposed approach changes the heuristic costs over time to account for expansion failures: (a) initial heuristic costs; (b) updated heuristic costs after the snake-like robot fails to pass through the opening at the top; and (c) updated heuristic costs after robot fails to pass through the opening on the right side. The color of each region r represents the shortest-path distance in the workspace decomposition from r to the goal (red: near, blue: far). Figures are best viewed in color and on screen.

3 Problem Formulation

A robot model is defined in terms of its state space S, control space U, and dynamics f. A state $s \in S$ defines the position, orientation, velocity, steering angle, and other components related to motion. A control $u \in U$ defines the external inputs that are used to control the robot. Dynamics are specified as a set of differential equations $\dot{s} = f(s, u)$ which indicate the changes that occur to the state s when applying the control u. An example is provided below.

Example: The snake-like robot used in the experiments is modeled as a car pulling several trailers. This provides a high-dimensional model with

second-order differential equations, defined as (adapted from [13, pp.731])

$$\dot{x} = v\cos(\theta_0)\cos(\psi) \qquad \dot{y} = v\sin(\theta_0)\cos(\psi) \qquad \dot{\theta}_0 = v\sin(\psi)/L$$
$$\dot{v} = u_a \qquad \dot{\psi} = u_\omega \qquad \dot{\theta}_i = \frac{v}{d}(\sin(\theta_{i-1}) - \sin(\theta_0)) \prod_{j=1}^{i-1} \cos(\theta_{j-1} - \theta_j) \tag{1}$$

The state $s = (x, y, \theta_0, v, \psi, \theta_1, \ldots, \theta_N)$ defines the position (x, y), orientation θ_0, velocity v, and steering angle ψ of the car, and the orientation θ_i of each of the N trailers. The control $u = (u_a, u_\omega)$ defines the acceleration and the rotational velocity of the steering angle. The body and hitch lengths (L and d) are set to small values so that the robot resembles a snake, as shown in Fig. 1.

Let W denote the workspace in which the robot operates. Let $O_1, \ldots, O_m \subset W$ denote the obstacles and let $G \subset W$ denote the goal region. A state $s \in S$ is considered valid if there are no self-collisions or collisions with obstacles when the robot is placed according to the position and orientation specified by s. Such function, $\textsc{collision} : S \rightarrow \{\texttt{true}, \texttt{false}\}$, can be implemented efficiently by using available collision-detection packages, such as PQP [12]. The goal is reached when the position defined by s, denoted by $\textsc{pos}(s)$, is in G. From a motion-planning perspective, the effect of the dynamics is captured by a function

$$s_{new} \leftarrow \textsc{motion}(f, s, u, dt), \tag{2}$$

which uses a fourth-order Runge-Kutta numerical integration to compute the new state s_{new} obtained by applying the control u to s for one time step dt. A motion trajectory $\zeta : \{0, 1, \ldots, \ell\} \rightarrow S$ is obtained by starting at a state s and applying a sequence of control inputs $\langle u_0, u_1, \ldots, u_{\ell-1} \rangle$ in succession, i.e.,

$$\zeta(0) = s \text{ and } \forall i \in \{0, \ldots, \ell - 1\} : \zeta(i + 1) = \textsc{motion}(f, \zeta(i), u_i, dt) \tag{3}$$

The motion-planning problem can now be stated as follows: Given a workspace W, obstacles $O_1, \ldots, O_m \subset W$, goal region $G \subset W$, robot model $\langle S, U, f \rangle$, and an initial state $s_{init} \in S$, compute a sequence of control inputs $\langle u_0, u_1, \ldots, u_{\ell-1} \rangle$ such that dynamically-feasible trajectory $\zeta : \{0, 1, \ldots, \ell\} \rightarrow S$ obtained by applying the control inputs in succession starting at s_{init} is collision-free and reaches the goal, i.e., $\forall i \in \{0, 1, \ldots, \ell\} : \textsc{collision}(\zeta(i)) = \texttt{false}$ and $\textsc{pos}(\zeta(\ell)) \in G$.

4 Method

Pseudocode for the approach is shown in Alg. 1. To facilitate presentation, the workspace decomposition is described first in Section 4.1. The overall guided search is described in Section 4.2.

4.1 Workspace Decomposition

To obtain a simplified planning layer, the unoccupied area of the workspace, i.e., $W \setminus (O_1 \cup \ldots \cup O_m \cup G)$, is decomposed into nonoverlapping triangular regions

Algorithm 1. Pseudocode for the proposed approach

Input: W: workspace; $O_1, \ldots, O_m \subset W$: obstacles; $G \subset W$: goal region
$\langle S, U, f \rangle$: robot model; $s_{init} \in S$: initial state; t_{max}: upper bound on runtime
Output: collision-free and dynamically-feasible trajectory to goal or `null` if no solution is found within t_{max} runtime

$D = (R, E, C) \leftarrow$ WorkspaceDecomposition$(W, O_1, \ldots, O_m, G)$
$\langle h(r_1), \ldots, h(r_n) \rangle \leftarrow$ HeuristicCosts(D, G)
$\langle \mathcal{T}, \Gamma \rangle \leftarrow$ InitMotionTree(s_{init})
while time $< t_{max}$ **do**
  for κ iterations **do**
    $\Gamma_r \leftarrow$ SelectGroup$(\Gamma, \langle h(r_1), \ldots, h(r_n) \rangle)$
    $\langle status, v_{last} \rangle \leftarrow$ ExpandMotionTree$(\mathcal{T}, \Gamma, \Gamma_r)$
    if $status =$ `goalReached` **then return** traj$(\mathcal{T}, v_{last})$
    if $status =$ `collisionEncountered` **then**
      $r_{next} \leftarrow$ first(ShortestPath$(D, region(v_{last}), G)$)
      for $r_{neigh} \in$ neighs(D, r_{next}) **do** IncreaseEdgeCost(D, r_{next}, r_{neigh})
  $\langle h(r_1), \ldots, h(r_n) \rangle \leftarrow$ UpdateHeuristicCosts(D, G)
return `null`

$t_1, \ldots, t_n$ [20]. Figs. 1–3 show some examples. The workspace decomposition is represented as an undirected, weighted, graph $D = (R, E, C)$, where R, E, C denote the regions, edges, and edge costs, respectively. The set of regions contains the triangles and the goal region, i.e., $R = \{t_1, \ldots, t_n, G\}$, since these regions can be used to generate collision-free motions. The set of edges captures the physical adjacency of the regions in the workspace decomposition, i.e.,

$$E = \{(r, r') : r, r' \in R \text{ share an edge}\}. \tag{4}$$

Edge costs are used to provide the approach with short routes to the goal region. Initially, the cost of an edge is set to the Euclidean distance between the centroids of the corresponding regions, i.e., $C(r, r') = ||centroid(r) - centroid(r')||_2$. As described later, the approach increases $C(r, r')$ when it fails to make progress expanding the motion tree from r to r'. These cost increases enable the approach to discover alternative routes to the goal. The approach also relies on a function LocateRegion : $W \rightarrow R \cup \{\perp\}$ which maps a point $p \in W$ to the region $r \in R$ that contains p. The symbol $\perp$ denotes the case when p falls inside an obstacle. LocateRegion is implemented efficiently to run in polylogarithmic time [3].

4.2 Guided Motion-Tree Search

The search for a collision-free and dynamically-feasible trajectory from the initial state s_{init} to the goal G is conducted by expanding a motion tree $\mathcal{T} = (V_{\mathcal{T}}, E_{\mathcal{T}})$. Each vertex $v \in V_{\mathcal{T}}$ is associated with a collision-free state, denoted by $state(v)$. Each edge $(v, v') \in E_{\mathcal{T}}$ is associated with some control input $u \in U$ such that $state(v') =$ motion$(f, state(v), u, dt)$. A solution is found when a vertex v that has reached G is added to $\mathcal{T}$, i.e., pos$(state(v)) \in G$. In that case, the solution

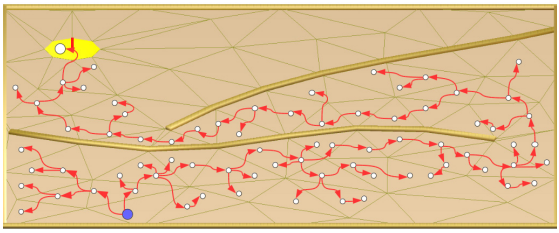


Fig. 2. Illustration of a motion tree. Initial state is shown as a blue circle; the other vertices are shown as white circles. Goal region is shown in yellow.

corresponds to the trajectory obtained by concatenating the motions associated with the edges in $\mathcal{T}$ from the root to v. An illustration is shown in Fig. 2.

To effectively guide the search, the motion-tree vertices are grouped together based on the corresponding regions in the workspace decomposition. In fact, for a region $r \in R$, let Γ_r denote all the vertices that map to r, i.e.,

$$\Gamma_r = \{v : v \in V_{\mathcal{T}} \wedge \textsc{pos}(state(v)) \in r\}. \tag{5}$$

This mapping induces a partition of the motion tree into groups, i.e.,

$$\Gamma = \{\Gamma_r : r \in R \wedge |\Gamma_r| > 0\}. \tag{6}$$

Since the decomposition $D = (R, E, C)$ provides a simplified planning layer, the shortest-path distance from r to G is used as a heuristic cost to estimate the feasibility of reaching G by expanding $\mathcal{T}$ from Γ_r. The search is then driven by (i) selecting a group Γ_r from Γ based on the heuristic costs, (ii) expanding $\mathcal{T}$ from Γ_r, and (iii) updating the heuristic costs to reflect the progress made. These steps are repeated until a solution is found or an upper bound on the runtime is reached. The rest of the section describes these procedures in more detail.

Group Selection. The group-selection strategy combines the heuristic costs with selection penalties in order to promote expansions from groups that are close to the goal or groups that have not been frequently explored in the past. Let $h(r)$ denote the shortest-path distance in $D = (R, E, C)$ from $r \in R$ to G. Let $nsel(\Gamma_r)$ denote the number of times Γ_r has been selected for expansion. The heuristic cost and the number of selections are combined to define a weight

$$w(\Gamma_r) = (\epsilon + 1 - h(r)/h_{max})^{\alpha} \beta^{nsel(\Gamma_r)}, \tag{7}$$

where $h_{max} = \max_{r' \in R} h(r')$, $\epsilon > 0$, $\alpha \geq 1$, and $0 < \beta < 1$. Among all the groups in Γ, the one with the maximum weight is selected for expansion, i.e.,

$$\textsc{SelectGroup}(\Gamma) = \arg\max_{\Gamma_r \in \Gamma} w(\Gamma_r). \tag{8}$$

Note that α serves to tune the strength of the heuristic by promoting selections of those groups that are close to the goal according to shortest-path distances in

Algorithm 2. ExpandMotionTree($\mathcal{T}, \Gamma, \Gamma_r$)

Input: $\mathcal{T}$: motion tree; Γ: partition of $\mathcal{T}$ into groups; Γ_r: group from which to expand $\mathcal{T}$

Output: Function attempts to add a collision-free and dynamically-feasible trajectory from a vertex in Γ_r. It returns $\langle \texttt{status}, v_{last} \rangle$ indicating the status of the expansion and the last vertex added to $\mathcal{T}$

1: $v \leftarrow$ select vertex at random from Γ_r
2: $r' \leftarrow$ select region at random from shortest path in $D = (R, E, C)$ from r to G
3: $p \leftarrow$ generate point at random inside r'
4: **for** several steps **do**
5: $u \leftarrow$ CONTROLLER($state(v), p$); $s_{new} \leftarrow$ MOTION($f, state(v), u, dt$)
6: **if** COLLISION(s_{new}) = `true` **then return** $\langle$`collision`$, v\rangle$
7: $v_{new} \leftarrow$ NewVertex(); $state(v_{new}) \leftarrow s_{new}$;
$region(v_{new}) \leftarrow$ LocateRegion(POS(s_{new})); $V_{\mathcal{T}} \leftarrow V_{\mathcal{T}} \cup \{v_{new}\}$;
8: $(v, v_{new}) \leftarrow$ NewEdge(); $control(v, v_{new}) \leftarrow u$; $E_{\mathcal{T}} \leftarrow E_{\mathcal{T}} \cup \{(v, v_{new})\}$
9: $\Gamma_{r_{new}} \leftarrow$ FIND($\Gamma, region(v_{new})$)
10: **if** $\Gamma_{r_{new}}$ = `null` **then** { $\Gamma_{r_{new}} \leftarrow$ NewGroup(r_{new}); INSERT($\Gamma, \Gamma_{r_{new}}$) }
11: INSERT($\Gamma_{r_{new}}, v_{new}$)
12: **if** $region(v_{new}) = G$ **then return** $\langle$`goalReached`$, v_{new}\rangle$
13: **if** NEAR(POS(s_{new}), p) = `true` **then return** $\langle$`targetReached`$, v_{new}\rangle$
14: $v \leftarrow v_{new}$
15: **return** $\langle$`targetNotReached`$, v\rangle$

$D = (R, E, C)$. To balance the greediness of the heuristic, β provides a penalty factor which reduces the weight each time Γ_r is selected for expansion. This guarantees that Γ_r will not always be selected, since, after a number of weight reductions, some other group will have larger weight than Γ_r. This enables the approach to avoid becoming stuck when expansions from Γ_r are infeasible due to constraints imposed by the obstacles and robot dynamics. Finally, a small nonzero value is used for ϵ to ensure that each Γ_r has a positive weight, which guarantees that every Γ_r will be eventually selected for expansion. This enables the approach to be methodical during the search.

Group Expansion. After selecting Γ_r, the objective is to expand $\mathcal{T}$ from a vertex $v \in \Gamma_r$ along the shortest path in $D = (R, E, C)$ from r to G. Pseudocode is shown in Alg. 2. The vertex v is selected uniformly at random from the vertices in Γ_r (Alg. 2:1). Random selections are commonly used in sampling-based motion planning as a way to promote expansions along different directions [5, chap. 7]. After selecting v, a target region r' is selected uniformly at random from the first few regions along the shortest path in D from r to G, and a target point p is generated uniformly at random inside r' (Alg. 2:2–3). This ensures that the target will not be too far away from r, which increases the likelihood of successful expansions. The objective is then to expand $\mathcal{T}$ from v toward p in order to get closer to G (Alg. 2:4–11). Specifically, a PID controller [2] is used to determine an input control u that would steer the robot toward p (Alg. 2:5). A new state, s_{new}, is obtained by integrating the motion equations f when the control u is applied

to $state(v)$ for one time step (Alg. 2:5). If s_{new} is in collision, the expansion from v terminates. Otherwise, a new vertex v_{new} and a new edge (v, v_{new}) are added to $\mathcal{T}$ (Alg. 2:6–8). At this time, the motion-tree partition is also updated. If $region(v_{new})$ had not been reached before, then a new group $\Gamma_{region(v_{new})}$ is created and added to Γ (Alg. 2:9–10). Otherwise, the group $\Gamma_{region(v_{new})}$ is retrieved from Γ. In each case, the vertex v_{new} is added to $\Gamma_{region(v_{new})}$ (Alg. 2:11). These updates give the approach the flexibility to expand the motion tree from new groups.

If s_{new} reached the goal region G, the motion-tree expansion terminates successfully (Alg. 2:12). The expansion from v also terminates if the target point p is reached (Alg. 2:113). Otherwise, the expansion continues from v_{new} (Alg. 2:14). In this way, ExpandMotionTree$(\mathcal{T}, \Gamma, \Gamma_r)$ expands the motion tree $\mathcal{T}$ from a vertex $v \in \Gamma_r$ along the shortest path in D from r to G.

Updating the Heuristic Costs. To counterbalance the greediness of the shortest-path heuristic, when a collision is encountered during the motion-tree expansion, the costs of the corresponding edges in the workspace decomposition $D = (R, E, C)$ are increased. More specifically, let v_{last} denote the last vertex that was added to $\mathcal{T}$ during the expansion from Γ_r (Alg. 1:6). Let r_{next} denote the first region of the shortest path in $D = (R, E, C)$ from $region(v_{last})$ to G (Alg. 1:8). The costs of the edges in D that have r_{next} as a vertex are then increased in order to reduce the likelihood of future expansions passing through r_{next} since the previous expansion resulted in a collision (Alg. 1:9). Specifically, the cost of each edge $(r_{next}, r_{neigh}) \in E$ is increased as

$$C(r_{next}, r_{neigh}) \leftarrow C(r_{next}, r_{neigh}) + \gamma / C(r_{next}, r_{neigh}), \tag{9}$$

where $\gamma > 0$. The intuition is that our estimation of the difficulty of passing through a region should increase when motion-tree expansions fail. The increase is large when $C(r_{next}, r_{neigh})$ is small, and it is slowly reduced over time as the edge cost becomes larger. The parameter γ can be tuned to provide appropriate cost increases based on the workspace size. As the experiments indicate, the approach works well for a wide range of parameter values and workspace sizes.

The heuristic costs $h(r_1), \ldots, h(r_n)$ are also updated in order to reflect the increases in the edge costs (Alg. 1:10). These updates enable the approach to discover alternative routes to reach G, as shown in Fig. 1. Note that $h(r_1), \ldots, h(r_n)$ can be computed by a single call to Dijkstra's shortest-path algorithm using G as the source. When dealing with large scenes, in order to save computation time, the heuristic costs can be updated after several attempts have been made to expand the motion tree instead of doing the updates after each attempt.

5 Experiments and Results

Experiments are conducted with a high-dimensional snake-like robot model whose dynamics are expressed by second-order differential equations, as described in Section 3. As shown in Figs. 1 and 3, the robot is required to

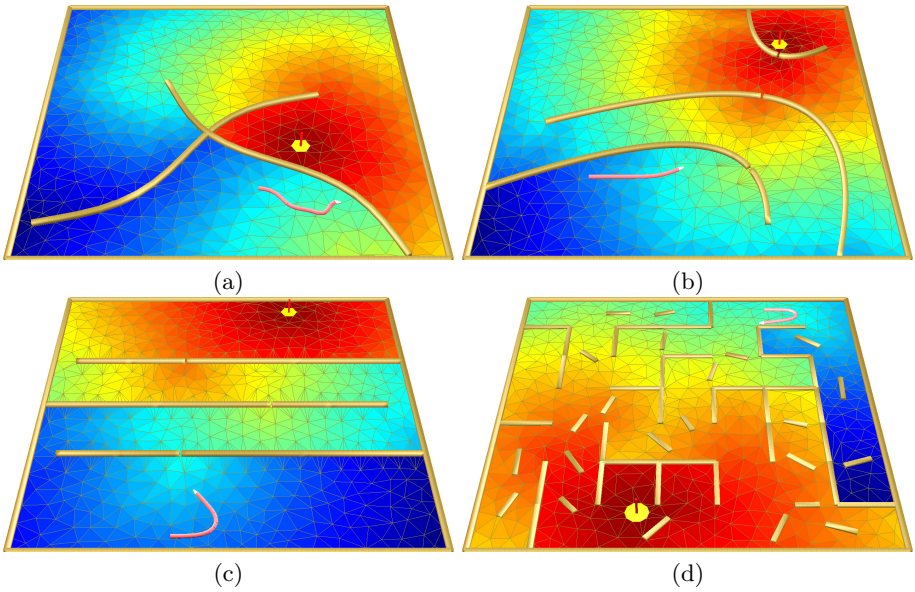


Fig. 3. The other four scenes used in the experiments (first scene is shown in Fig. 1). The color of each region r represents the distance of the shortest path in the workspace decomposition from r to the goal (red: near, blue: far). Videos showing solutions obtained by the approach are available for download [21].

move in complex scenes characterized by narrow passages, some of which are so small that it is impossible for the robot to pass through them. These scenes demonstrate the ability of the approach to quickly discover alternative routes to the goal. Experiments are also conducted with a scene which does not have impassible narrow passages (Fig. 3(d)) to show that the approach works well in different scenarios.

The approach is compared to RRT [14], which is one of the most popular sampling-based motion planners. As recommended, the RRT implementation uses the connect version, goal bias, and efficient data structures for nearest neighbors [4]. The approach is also compared to prior work on coupling motion planning with discrete search [18], using a highly efficient implementation.

Experiments are also conducted to measure the scalability of the approach when varying the dimensionality of the robot state (by changing the number of links) or the size of the scene. Results are also presented to demonstrate that the approach works well for a wide range of parameter values.

Due to the randomized nature of sampling-based motion planning, each method is run 60 times for each scenario. A time limit of 100s is set for each run. Running time measures everything from reading the input until finding a solution. Results show the mean runtime and standard deviation after discarding the five best and worst runs in order to reduce the influence of outliers. Experiments were run on an Intel Core i7 machine using Fedora 20 and GNU g++-4.9.2.

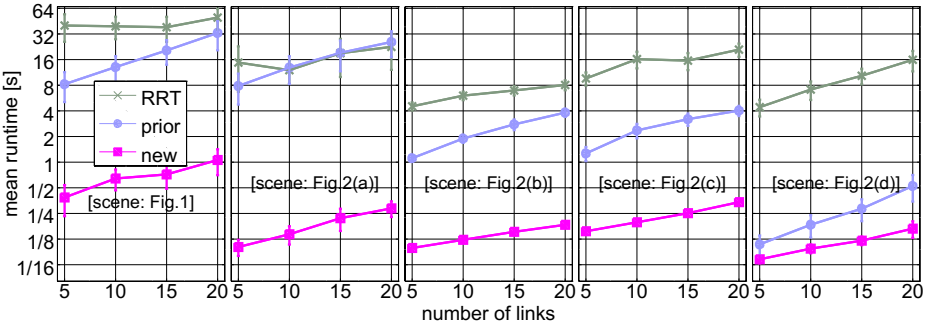


Fig. 4. Runtime results when varying the number of links of the snake-like robot model. Each bar indicates one standard deviation. Due to the significant differences in runtime, logscale is used for the y-axis with the label showing the actual value rather than its logarithm. New refers to the proposed approach and prior refers to prior work [18].

Results on Problem Dimensionality. Fig. 4 provides a summary of the results when increasing the number of links in the snake-like robot model. The results indicate that the proposed approach significantly outperforms RRT and the prior work [18]. RRT is known to have difficulties when dealing with narrow passages. In fact, since RRT expands the motion tree from the nearest neighbor to a randomly-sampled state, RRT often becomes stuck attempting to expand from vertices that are close to obstacles. The motion planner from prior work [18], although significantly faster than RRT, still has difficulty solving the problems with multiple narrow passages, some of which are impassible. The prior planner spends considerable time before realizing that it should consider alternative routes to the goal. In contrast, the proposed approach increases the edge costs of the workspace decomposition when it fails to make progress from one region to the next. These cost increases enable the approach to update the shortest-path heuristics and to quickly discover alternative routes to the goal.

Results on Scene Size. Table 1 shows the results when varying the scene size. During the scaling, the dimensions of the narrow passages are kept the same as in the original scene. In this way, the difficulty of the problem due to narrow passages remains the same in the scaled versions as in the original scene. When considering larger scenes, an increase in runtime is expected since longer trajectories have to be planned. Results in Table 1 show that RRT quickly times out in the large scenes. There is also a significant increase in the runtime of the prior planner [18] since it now spends more time before realizing that it should consider alternative routes. The running time of the proposed approach also increases, but even in the large scenes it remains computationally efficient.

Parameter Selection. Table 2 summarizes the impact of the parameter selection on the performance of the proposed approach. The results indicate that the approach works well for a wide selection of parameter values. When starting

Table 1. Mean runtime (std in parentheses) when scaling the scene. During scaling, the sizes of the narrow passages are kept the same as in the original scene. Entries marked with X indicate failure. Results are shown for the scene in Fig. 3(c) where the snake-like robot model has 15 links.

scene scaling factor	0.75	1.0	2.0	3.0	4.0
new	0.18s (0.03)	0.25s (0.05)	0.93s (0.12)	3.38s (0.30)	7.60s (0.69)
prior	1.52s (0.93)	3.20s (1.06)	32.68s (17.02)	97.34 (5.10)	X
RRT	7.20s (2.93)	15.56s (7.08)	95.14 (9.30)	X	X

Table 2. Results when varying the parameters of the approach. Results along each row are obtained by changing the corresponding parameter as indicated and keeping the other parameters to their default values (shown in bold). Recall that α tunes the strength of the heuristic (Eqn. 7), β provides the group-selection penalty (Eqn. 7), γ impacts the increase in the edge costs (Eqn. 9), and κ indicates the frequency of updating the heuristic costs (Alg. 1:5). Results are shown for the scene in Fig. 3(b) where the snake-like robot model has 15 links.

param	param values					mean runtime[s] (std)				
α	2	4	**8**	12	16	0.18 (0.02)	0.15 (0.02)	0.15 (0.02)	0.16 (0.02)	0.15 (0.02)
β	0.5	0.65	0.75	**0.85**	0.95	0.18 (0.02)	0.17 (0.02)	0.16 (0.02)	0.16 (0.02)	0.15 (0.02)
γ	15	25	**50**	75	100	0.17 (0.02)	0.15 (0.02)	0.15 (0.02)	0.15 (0.02)	0.15 (0.02)
κ	1	5	10	**20**	30	0.99 (0.17)	0.28 (0.05)	0.19 (0.03)	0.15 (0.02)	0.14 (0.02)

with a new problem, our recommendation is to use the default values shown in Table 2 as they have worked well for a variety of problems.

6 Discussion

This paper presented an efficient approach for planning dynamically-feasible trajectories that enable a mobile robot to reach a goal region while avoiding collisions with obstacles. The approach used a workspace decomposition to partition a motion tree into equivalent groups and guide the expansion of the motion tree along shortest-path routes. A key component of the approach was its adjustment of the edge costs in order to quickly discover alternative routes to the goal when expansions along the current route failed to make progress. Comparisons to related work showed significant speedups. Directions for future work include further improvements to the interplay between sampling-based motion planning and discrete search, investigation of machine-learning techniques to find optimal parameter values, and extensions of the approach to multiple robots.

Acknowledgement. This work was supported by NSF IIS-1449505 and NSF ACI-1440587.

References

1. Alterovitz, R., Goldberg, K.: Motion Planning in Medicine: Optimization and Simulation Algorithms for Image-Guided Procedures. Springer Tracts in Advanced Robotics (2008)
2. Åström, K.J., Hägglund, T.: PID controllers: theory, design, and tuning. The Instrumentation, Systems, and Automation Society (1995)
3. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.H.: Computational Geometry: Algorithms and Applications, 3rd edn. Springer-Verlag (2008)
4. Brin, S.: Near neighbor search in large metric spaces. In: International Conference on Very Large Data Bases, pp. 574–584 (1995)
5. Choset, H., Lynch, K.M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L.E., Thrun, S.: Principles of Robot Motion: Theory, Algorithms, and Implementations. MIT Press (2005)
6. Şucan, I.A., Kavraki, L.E.: A sampling-based tree planner for systems with complex dynamics. IEEE Transactions on Robotics **28**(1), 116–131 (2012)
7. Dalibard, S., Laumond, J.-P.: Control of probabilistic diffusion in motion planning. In: Chirikjian, G.S., Choset, H., Morales, M., Murphey, T. (eds.) Algorithmic Foundation of Robotics VIII. STAR, vol. 57, pp. 467–481. Springer, Heidelberg (2009)
8. Devaurs, D., Simeon, T., Cortés, J.: Enhancing the transition-based RRT to deal with complex cost spaces. In: IEEE International Conference on Robotics and Automation, pp. 4120–4125 (2013)
9. Gipson, B., Moll, M., Kavraki, L.E.: Resolution independent density estimation for motion planning in high-dimensional spaces. In: IEEE International Conference on Robotics and Automation, pp. 2437–2443 (2013)
10. Hsu, D., Kindel, R., Latombe, J.C., Rock, S.: Randomized kinodynamic motion planning with moving obstacles. International Journal of Robotics Research **21**(3), 233–255 (2002)
11. Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. International Journal of Robotics Research **30**(7), 846–894 (2011)
12. Larsen, E., Gottschalk, S., Lin, M., Manocha, D.: Fast proximity queries with swept sphere volumes. In: IEEE International Conference on Robotics and Automation, pp. 3719–3726 (2000)
13. LaValle, S.M.: Planning Algorithms. Cambridge University Press (2006)
14. LaValle, S.M.: Motion planning: The essentials. IEEE Robotics & Automation Magazine **18**(1), 79–89 (2011)
15. LaValle, S.M., Kuffner, J.J.: Randomized kinodynamic planning. International Journal of Robotics Research **20**(5), 378–400 (2001)
16. Le, D., Plaku, E.: Guiding sampling-based tree search for motion planning with dynamics via probabilistic roadmap abstractions. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 212–217 (2014)
17. Plaku, E., Kavraki, L.E., Vardi, M.Y.: Impact of workspace decompositions on discrete search leading continuous exploration (DSLX) motion planning. In: IEEE International Conference on Robotics and Automation, pp. 3751–3756 (2008)
18. Plaku, E.: Robot motion planning with dynamics as hybrid search. In: AAAI Conference on Artificial Intelligence, pp. 1415–1421 (2013)
19. Plaku, E., Kavraki, L.E., Vardi, M.Y.: Motion planning with dynamics by a synergistic combination of layers of planning. IEEE Transactions on Robotics **26**(3), 469–482 (2010)

20. Shewchuk, J.R.: Delaunay refinement algorithms for triangular mesh generation. Computational Geometry: Theory and Applications **22**(1–3), 21–74 (2002). http://www.cs.cmu.edu/~quake/triangle.html
21. Wells, A., Plaku, E.: Supplementary material, http://faculty.cua.edu/plaku/TAROS15Videos.zip