Efficient Collision Checking in Sampling-based Motion Planning

Joshua Bialkowski, Sertac Karaman, Michael Otte, and Emilio Frazzoli

Abstract Collision checking is generally considered to be the primary computational bottleneck in sampling-based motion planning algorithms. We show that this does not have to be the case. More specifically, we introduce a novel way of implementing collision checking in the context of sampling-based motion planning, such that the amortized complexity of collision checking is negligible with respect to that of the other components of sampling-based motion planning algorithms. Our method works by storing a lower bound on the distance to the nearest obstacle of each normally collision-checked point. New samples may immediately be determined collision free—without a call to the collision-checking procedure—if they are closer to a previously collision-checked point than the latter is to an obstacle. A similar criterion can also be used to detect points inside of obstacles (i.e., points that are in collision with obstacles). Analysis proves that the expected fraction of points that require a call to the normal (expensive) collision-checking procedure approaches zero as the total number of points increases. Experiments, in which the proposed idea is used in conjunction with the RRT and RRT* path planning algorithms, also validate that our method enables significant benefits in practice.

1 Introduction

Sampling-based algorithms are a popular and general approach for solving high-dimensional motion planning problems in robotics, computer graphics, and synthetic biology [1, 6, 10]. The main idea is to construct collision-free trajectories by joining points sampled from the state space, thus avoiding computations based on an explicit representation of the obstacles. The main components of sampling-based

Joshua Bialkowski, Sertac Karaman, Michael Otte, and Emilio Frazzoli
Massachussetts Institute of Technology, 77 Massachussetts Ave., Cambridge MA 02139,
Joshua Bialkowski e-mail: jbialk@mit.edu, Sertac Karaman e-mail: sertac@mit.edu,
Michael Otte e-mail: ottemw@mit.edu, Emilio Frazzoli e-mail: frazzoli@mit.edu.

algorithms are: (i) a sampling scheme; (ii) a collision-checking function that determines if a point is in collision, given an obstacle set; (iii) a proximity search function that returns "neighbors" to a point, given a point set; and (iv) a local planning function that returns a trajectory between two given points.

The vast body of literature on sampling-based motion planning contains many algorithms that implement the aforementioned four components in different ways. Arguably, the main paradigms are the Probabilistic RoadMap (PRM) [9] and the Rapidly-exploring Random Tree (RRT) [12] algorithms, aimed respectively to multi- and single-query problems. Both PRM and RRT are known to be probabilistically complete, i.e., if a (robust) solution exists, then a solution will almost surely be found as the number of samples increases. Other algorithms use variations on PRM and RRT to improve performance, e.g., [7, 5] bias sampling based on collision-checking results, [4] delay collision checks until needed (LazyPRM), and [8] provide asymptotic optimality guarantees on path quality (PRM* and RRT*).

Collision checking is widely considered the primary computational bottleneck of sampling-based motion planning algorithms (e.g., see [11]). Our main contribution is to show that this does not have to be the case. We introduce a novel collision checking implementation that has negligible amortized complexity vs. the proximity searches that are already at the core of sampling-based motion planning. As a consequence, proximity searches are identified as the main determinant of complexity in sampling-based motion planning algorithms, rather than collision checking.

Traditionally, collision checking is handled by passing a point query to a "Boolean black box" collision checker that returns either true or false depending on if the point is in collision with obstacles or not, respectively. In this paper, we place a stronger requirement on the collision-checking procedure, which is now assumed to return a lower bound on the minimum distance to the obstacle set. Although computing such a bound is harder than checking whether a point is in collision or not, leveraging this extra information allows us to skip explicit collision checks for a large fraction of the subsequent samples. Indeed, the probability of calling the explicit collision checking procedure, and hence the amortized computational complexity due to collision checking, converges to zero as the number of samples increases.

The rest of this paper is organized as follows: In Section 2 we introduce notation and state our problem and main results. In Section 3 we present our new collision checking method. In Section 4 we analyze the expected fraction of samples that will require a collision distance query and calculate the expected runtime savings of the proposed approach (e.g., for RRT, PRM, and many of their variants). In Section 5 we demonstrate performance improvement when used with RRT and RRT*. In Section 6 we draw conclusions and discuss directions for future work.

2 Notation, Problem Statement, and Main Results

Let $X = (0,1)^d$ be the *configuration space*, where $d \in \mathbb{N}$, $d \geq 2$ is the dimension of the space. Let X_{obs} be the *obstacle region*, such that $X \setminus X_{\text{obs}}$ is an open set, and

denote the *obstacle-free space* as $X_{\text{free}} = \text{cl}(X \setminus X_{\text{obs}})$, where $\text{cl}(\cdot)$ denotes the closure of a set. The *initial condition* x_{init} is an element of X_{free}, and the *goal region* X_{goal} is an open subset of X_{free}. A path planning problem is defined by a triplet $(X_{\text{free}}, x_{\text{init}}, X_{\text{goal}})$. For the sake of our discussion, we will assume that sampling-based algorithms take as input a path planning problem $(X_{\text{free}}, x_{\text{init}}, X_{\text{goal}})$, and an integer $n \in \mathbb{N}$, and return a graph $G_n = (V_n, E_n)$, where $V_n \subset X_{\text{free}}$, $\text{card}\,(V_n) \leq n$, and $E_n \subset V_n \times V_n$. The solution of the path planning problem (if found) can be recovered from such a graph, e.g., using standard shortest-path algorithms. Sampling-based algorithms map a sequence of sampled points ω to a sequence of graphs. Let $G_n(\omega) = (V_n(\omega), E_n(\omega))$ denote the output of the algorithm, given ω.

The problem we address in this paper is how to reduce the complexity of sampling-based motion planning algorithms, i.e., the (expected) number of operations needed to compute G_n. In particular, we consider incremental complexity—the expected difference in the total number of operations needed to compute $G_{n+1}(\omega)$ and $G_n(\omega)$. In the sequel, we characterize complexity based on the number of samples n and the environment description X_{obs}, while keeping the dimension d of the space a constant. For convenience, we assume stochastic sampling that draws uniformly and independently from X. Our results extend to deterministic and/or non-uniform sampling procedures, as long as the sampling procedure satisfy the technical conditions required for probabilistic or resolution completeness [11].

The computational complexity of a sampling-based algorithm can be decomposed in terms of the complexity of its primitive operations. The complexity of drawing a sample is bounded by a constant c_{sample}. The complexity of a collision check is bounded by $c_{\text{cc}}(X_{\text{obs}})$ a function that depends only on the description of the environment—note that the expected complexity of checking collisions with n_{obs} convex obstacles is $O(\log(n_{\text{obs}})^d)$ [13]. The complexity of (approximate) proximity searches that return the $O(\log n)$ nearest elements to a query point from a set of cardinality n is $O(\log n)$ [13]. The latter applies to k-nearest neighbor searches (k fixed or scaling as $\log n$), and range searches among uniform random samples for those points within a ball of volume scaling as $\log(n)/n$.

The complexity of the local planner is bounded by a constant c_{plan} that does not depend on the environment description. For convenience we also include in c_{plan} all other constant time bookkeeping operations (cost updates, graph rewiring, etc.) that are performed for each new edge. Each candidate path generated by the local planner must also be checked for collisions, with complexity bounded by $c_{\text{path}}(X_{\text{obs}})$ a function that depends only on the description of the environment.

The expected incremental complexity of sampling-based motion planning algorithms can be evaluated by examining the work induced by each new sample:

- RRT: Find the nearest neighbor, generate a new point, check the new point for collision, and connect the new point to the nearest neighbor.
- k-PRM: Collision check the sample, find and connect to its k-nearest neighbors.
- RRT*, PRM*: Collision check the sample, find the neighbors within a ball of volume scaling as $\log(n)/n$—or, equivalently, the $(k \log n)$-nearest neighbors, for a fixed k—and connect to them.

Algorithm	Proximity Search	Point Collision Checking	Local Planning	Path Collision Checking
RRT	$O(\log n)$	$c_{\text{cc}}(X_{\text{obs}})$	c_{plan}	$c_{\text{path}}(X_{\text{obs}})$
k-PRM	$O(\log n)$	$c_{\text{cc}}(X_{\text{obs}})$	$k\,c_{\text{plan}}$	$k\,c_{\text{path}}(X_{\text{obs}})$
RRT*, PRM*	$O(\log n)$	$c_{\text{cc}}(X_{\text{obs}})$	$O(\log n)\,c_{\text{plan}}$	$O(\log n)\,c_{\text{path}}(X_{\text{obs}})$

Table 1: Asymptotic bounds on the expected incremental complexity of some standard sampling-based motion planning algorithms.

Table 1 summarizes the main contributions to the expected incremental complexity of these algorithms, in their standard implementation, based on collision-checking queries. Note that the local planning and path checking only occur if the new sample is not in collision with obstacles.

It is important to note that the asymptotic bounds may be misleading. In practice, the cost of collision checking the new sample and a new candidate connection $c_{\text{cc}}(X_{\text{obs}}) + c_{\text{path}}(X_{\text{obs}})$ often effectively dominates the incremental complexity, even for very large n. Hence, collision checking is typically regarded as the computational bottleneck for sampling-based motion planning algorithms. The cost of collision checking is even more evident for algorithms such as RRT* and PRM*, where collision checks are carried out for each of the $O(\log n)$ candidate connections at each iteration. Moreover, the ratio of the incremental complexity of RRT* to that of RRT depends on the environment via $c_{\text{path}}(X_{\text{obs}})$, and is potentially large, although constant with respect to n.

In our approach, we replace standard collision checks with approximate minimum-distance computations. The complexity of computing a non-trivial lower bound on the minimum distance of a point from the obstacle set is also bounded by a function that depends on the description of the environment. In particular, we will require our collision checking procedure to return a lower bound $\bar{d}$ on the minimum distance d^* that satisfies $\alpha d^* \leq \bar{d} \leq d^*$, for some $\alpha \in (0,1]$. The computational cost of such a procedure will be indicated with $c_{\text{dist}}(X_{\text{obs}})$. While (approximate) distance queries are more expensive than collision checking, the ratio $c_{\text{dist}}(X_{\text{obs}})/c_{\text{cc}}(X_{\text{obs}})$ is bounded by a constant, independent of n.

The output of the minimum-distance computations is stored in a data structure that augments the standard graph constructed by sampling-based algorithms. Using this information, it is possible to reduce the number of explicit collision checks and minimum-distance computations that need to be performed. As we will later show, using our approach to modify the standard algorithms results in the asymptotic bounds on expected incremental complexity summarized in Table 2, where p_{cc} is a function such that $\limsup_{n\to\infty} p_{\text{cc}}(n) = 0$. Again, the local planning and path checking only occur if the new sample is not in collision with obstacles.

Inspection of table 2 reveals that the incremental complexity of our version of RRT and k-PRM is less sensitive to the cost of collision checking. Moreover, the asymptotic ratio of the incremental complexity of our version of RRT* and RRT is a constant that does not depend on the environment.

Mod. Algorithm	Prox. Search	Collision Checking	Local Planning	Path Collision Checking
RRT	$O(\log n)$	$O(\log n) + p_{\text{cc}}(n) c_{\text{cc}}(X_{\text{obs}})$	c_{plan}	$p_{\text{cc}}(n) c_{\text{path}}(X_{\text{obs}})$
k-PRM	$O(\log n)$	$O(\log n) + p_{\text{cc}}(n)\, c_{\text{cc}}(X_{\text{obs}})$	$k c_{\text{plan}}$	$k\, p_{\text{cc}}(n)\, c_{\text{path}}(X_{\text{obs}})$
RRT*, PRM*	$O(\log n)$	$O(\log n) + p_{\text{cc}}(n)\, c_{\text{cc}}(X_{\text{obs}})$	$O(\log n) c_{\text{plan}}$	$O(\log n)\, p_{\text{cc}}(n)\, c_{\text{path}}(X_{\text{obs}})$

Table 2: Asymptotic bounds on the expected incremental complexity of some sampling-based motion planning algorithms, modified according to the proposed approach; p_{cc} is a function, such that $\limsup_{n\to\infty} p_{\text{cc}}(n) = 0$

3 Proposed Algorithm

Before describing our proposed modification to the collision checking procedure in Section 3.1, we first formalize its primitive procedures and data structure.

Sampling: Let $\texttt{Sample} : \omega \mapsto \{\texttt{Sample}_i(\omega)\}_{i\in\mathbb{N}} \subset X$ be such that the random variables $\texttt{Sample}_i$, $i \in \mathbb{N}$, are independent and identically distributed (i.i.d.). The samples are assumed to be from a uniform distribution, but results extend naturally to any absolutely continuous distribution with density bounded away from zero on X.

Nearest Neighbors: Given a finite point set $S \subset X$ and a point $x \in X$, the function $\texttt{Nearest} : (S,x) \mapsto s \in S$ returns the point in S that is closest to x,

$$\texttt{Nearest}(S,x) := \operatorname*{argmin}_{s\in S} \|x-s\|,$$

where $\|\cdot\|$ is a metric (e.g., Euclidean distance—see [12] for alternative choices). A set-valued version is also considered, $\texttt{kNearest} : (S,x,k) \mapsto \{s_1, s_2, \ldots, s_k\}$, returns the k vertices in S that are nearest to x with respect to $\|\cdot\|$. By convention, if the cardinality of S is less than k, then the function returns S.

Near Vertices: Given a finite point set $S \subset X$, a point $x \in X$, and a positive real number $r \in \mathbb{R}_{>0}$, the function $\texttt{Near} : (S,x,r) \mapsto S_{\text{near}} \subseteq S$ returns the vertices in S that are inside a ball of radius r centered at x,

$$\texttt{Near}(S,x,r) := \{s \in S : \|s-x\| \le r\}.$$

Set distance: a closed set $S \subset X$ and a point $x \in X$, `SetDistance` returns a non-trivial lower bound on the minimum distance from x to S, i.e., for some $\alpha \in (0,1]$,

$$\alpha \min_{s\in S} \|s-x\| \le \texttt{SetDistance}(S,x) \le \min_{s\in S} \|s-x\|.$$

Segment Collision Test: Given points $x, y \in X$, $\texttt{CFreePath}(x,y)$ returns `True` if the line segment between x and y lies in X_{free}, i.e., $[x,y] \subset X_{\text{free}}$, and `False` otherwise.

Data Structure: We achieve efficient collision checking by storing additional information in the graph data structure that is already used by most sampling-based algorithms. Specifically, we use the "augmented graph" $AG = (V, E, C_{\text{free}}, C_{\text{obs}}, \text{Dist})$, where $V \subset X_{\text{free}}$ and $E \subset V \times V$ are the usual vertex and edge sets.The sets

$C_{\text{free}} \subset X_{\text{free}}$ and $C_{\text{obs}} \subset X_{\text{obs}}$ are sets of points for which an explicit collision check has been made. For points in C_{free} or C_{obs} the map $\text{Dist} : C_{\text{free}} \cup C_{\text{obs}} \to \mathbb{R}_{\geq 0}$ stores the approximate minimum distance to the obstacle set or to the free space, respectively. The vertices V and edges E are initialized according to the particular sampling-based algorithm in use. The sets C_{free} and C_{obs} are initialized as empty sets.

3.1 Modified Collision Checking Procedures

Using the augmented graph data structure allows us to modify the collision checking procedures as shown in Algorithms 1 and 2, respectively for points and paths. For convenience, we assume that the augmented data structure AG can be accessed directly by the collision checking procedures, and do not list it as an input.

Point Collision Test: Given a point $x \in X$, the function `CFreePoint`(x) returns `True` if $x \in X_{\text{free}}$, and `False` otherwise. When a new sample q is checked for collision, we first check if we can quickly determine whether it is in collision or not using previously computed information, lines 1.1-1.6. In particular, we use the vertex $v_{\text{near}} \in C_{\text{free}}$ that is nearest to q (found on line 1.1). If q is closer to v_{near} than the v_{near} is to an obstacle, then clearly q is collision free, lines 1.3-1.4. Otherwise, we find the vertex $o_{\text{near}} \in C_{\text{obs}}$ that is nearest to q, line 1.2. If q is closer to o_{near} than o_{near} is to the free space, then clearly q is in collision, line 1.5-1.6.

If these two checks prove inconclusive, then a full collision check is performed using set distance computations. First, one can compute the approximate minimum distances from q to the obstacle set, and to the free set, respectively indicated with d_{obs} and d_{free}, lines 1.7-1.8. If $d_{\text{obs}} > 0$, q is in X_{free} and is added to the set C_{free} of vertices that have been explicitly determined to be collision-free, lines 1.9-1.10. Furthermore, $\text{Dist}(q)$ is set to be equal to d_{obs}, line 1.11. Otherwise, q is in X_{obs}, and is added to the set C_{obs}, with $\text{Dist}(q) = d_{\text{free}}$, lines 1.13-1.15.

Path Set Collision Test: Given a vertex set $S \subseteq V$ and point x, the function `BatchCFreePath`(S,x) returns a set of edges $H \subseteq (V \cup \{x\}) \times (V \cup \{x\})$ such that for all $h = (x,y) \in H$, `CollisionFreePath`(x,y) evaluates to `True`. The form of S depends on the particular planning algorithm that is being used, e.g.,

- RRT: $S = \{\texttt{Nearest}(V,x)\}$ is a singleton containing the nearest point to x in V.
- k-PRM: $S = \texttt{kNearest}(V,x,k)$ contains up to k nearest neighbors to x in V.
- RRT* and PRM*: S contains $O(\log n)$ points—either the points within a ball of volume scaling as $\log(n)/n$ centered at x, or the $(k \log N)$-nearest neighbors to x.

We assume `CFreePoint`(x) is called prior to `BatchCFreePath`(S,x), and thus x is collision-free. For each pair (s,x), $s \in S$, the first step is to check whether the segment $[s,x]$ can be declared collision-free using only the information already available in AG, lines 2.2-2.5. Let $v_{\text{near}} \in C_{\text{free}}$ be the vertex in C_{free} that is nearest to x, line 2.2. If both s and x are closer to v_{near} than v_{near} is to the obstacle set, then clearly the segment $[s,x]$ is collision free, lines 2.4-2.5 ($\|x - v_{\text{near}}\| \leq \text{Dist}(v_{\text{near}})$

Algorithm 1: `CFreePoint`(q)

```
1  v_near ← Nearest(C_free, q);
2  o_near ← Nearest(C_obs, q);
3  if ||q − v_near|| ≤ Dist(v_near) then
4  |   return True
5  else if ||q − o_near|| ≤ Dist(o_near) then
6  |   return False
7  d_obs ← SetDistance(X_obs, q) ;
8  d_free ← SetDistance(X_free, q) ;
9  if d_obs > 0 then
10 |   C_free ← C_free ∪ {q};
11 |   Dist(q) ← d_obs;
12 |   return True
13 else
14 |   C_obs ← C_obs ∪ {q};
15 |   Dist(q) ← d_free;
16 |   return False
```

Algorithm 2: `BatchCFreePath`(S,x)

```
1  H ← ∅;
2  v_near ← Nearest(C_free, x);
3  foreach s ∈ S do
4  |   if ||s − v_near|| ≤ Dist(v_near) then
5  |   |   H ← H ∪ {(s,x)};
6  |   else if CFreePath(s,x) then
7  |   |   H ← H ∪ {(s,x)};
8  return H
```

Algorithm 3: `ApproxCDistPolytope`(A,c,q)

```
1  d ← (Aq − c) ;
2  d_min = −||−d||_∞ ;
3  d_max = ||d||_∞ ;
4  if d_max > 0 then
5  |   return (False, d_max) ;
6  else
7  |   return (True, −d_min) ;
```

is automatic, given that `CFreePoint`(x) has already been called). If this check is inconclusive, then a full collision check is performed, lines 2.6-2.7.

Approximate set collision distance: The set distance computation method depends on the choice of an obstacle index. For two dimensional polygonal obstacles, a segment Voronoi diagram is an index for which set distance computation is efficient and exact. For obstacles which are polytopes represented as the set $\{x : Ax \leq c\}$, Algorithm 3 is sufficient for calculating a lower bound on the set distance.

Efficient collision checking: We now illustrate how to improve the efficiency of standard sampling-based motion planning algorithms. As an example, Algorithms 4 and 5 show modified versions of the RRT and PRM* (pseudo-code based on [8]). *AG* is initialized on lines 4.1/5.1. Standard point collision checks are replaced with `CFreePoint` (i.e., Algorithm 1), lines 4.4/5.4. The code generating neighbor connections from a new sample is modified to generate those connections in a batch manner via a single call to `BatchCFreePath` (i.e., Algorithm 2), lines 4.6/5.7.

4 Analysis

Suppose *ALG* is a sampling-based motion planning algorithm. Let $I_{\text{point}}(n)$ denote the indicator random variable for the event that the `SetDistance` procedure in Algorithm 1 is called by *ALG* in the iteration in which the n-th sample is drawn. Similarly, define $I_{\text{path}}(n)$ as the indicator random variable for the event that the `CFreePath` procedure in Algorithm 2 is called by *ALG* in the iteration in which the

Algorithm 4: Modified RRT

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$; $C_{\text{free}} \leftarrow \emptyset$; $C_{\text{obs}} \leftarrow \emptyset$;
for $i = 1, \ldots, n-1$ **do**
  $x_{\text{rand}} \leftarrow \texttt{Sample}_i(\omega)$;
  if $\texttt{CFreePoint}(x_{\text{rand}})$ **then**
    $x_{\text{nearest}} \leftarrow \texttt{Nearest}(V, x_{\text{rand}})$;
    $H \leftarrow \texttt{BatchCFreePath}(\{x_{\text{nearest}}\}, x_{\text{rand}})$;
    if $H \neq \emptyset$ **then**
      $V \leftarrow V \cup x_{\text{rand}}$;
      $E \leftarrow E \cup H$;
return $G = (V, E)$;

Algorithm 5: Modified PRM*

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$; $C_{\text{free}} \leftarrow \emptyset$; $C_{\text{obs}} \leftarrow \emptyset$;
for $i = 1, \ldots, n-1$ **do**
  $x_{\text{rand}} \leftarrow \texttt{Sample}_i(\omega)$;
  if $\texttt{CFreePoint}(x_{\text{rand}})$ **then**
    $V \leftarrow V \cup \{x_{\text{rand}}\}$;
foreach $v \in V$ **do**
  $S \leftarrow \texttt{Near}(V, v, \gamma_{\text{PRM}}(\log(n)/n)^{1/d})$;
  $H \leftarrow \texttt{BatchCFreePath}(S \setminus \{v\}, v)$;
  $E \leftarrow E \cup H$;
return $G = (V, E)$;

n-th sample is drawn. Define $I(n) := I_{\text{point}}(n) + I_{\text{path}}(n)$, and let $p_{\text{cc}}(n) = \mathbb{E}[I(n)]$. We prove our main result for algorithms satisfying the following assumptions:

(A1) *ALG* calls the `CFreePoint` procedure $O(1)$ times per iteration;
(A2) At the n-th iteration, *ALG* calls the batch `CFreePath` procedure with $O(\log n)$ paths, and all such paths lie inside a ball of radius $o(n^{-1/(d+1)})$ as $n \to \infty$.

Our main result is stated in the following theorem.

Theorem 1. *Suppose ALG satisfies Assumptions (A1) and (A2). Then ALG, implemented using the proposed collision-checking algorithm, has zero asymptotic expected incremental set-distance complexity, i.e.,*

$$\limsup_{n \to \infty} p_{\text{cc}}(n) = 0.$$

Note that all algorithms in Table 2 satisfy Assumptions (A1) and (A2). Hence, the results summarized in Table 2 can be deduced from Theorem 1.

The rest of this section is devoted to the proof of Theorem 1. The key idea behind the proof is to divide up the state space into several cells. This particular discretization allows us to compute a bound on the expected number of samples that require Algorithm 1 to call the set-distance procedure. We show that this number grows slower than n, which implies Theorem 1.

Consider a partitioning of the environment into cells, where the size and number of cells used in the partition grows with the number of samples n. More precisely, divide up the configuration space $[0,1]^d$ into openly-disjoint hypercube cells with edge length $l_n := n^{-\frac{1}{d+1}}$. Given $\mathbf{z} \in \mathbb{Z}^d$, we define the cell with index $\mathbf{z}$ as the L^∞ ball of radius l_n centered at $l_n \mathbf{z}$, i.e., $C_n(\mathbf{z}) := \left\{x' \in X : \|x' - l_n \mathbf{z}\|_\infty \leq l_n\right\}$, where $l_n \mathbf{z}$ is the scalar-vector multiplication of $\mathbf{z}$ by l_n, and $\|\cdot\|_\infty$ is the infinity norm. Let $\mathbf{Z}_n \subset \mathbb{Z}^d$ denote the smallest set for which $\mathbf{C}_n := \{C_n(\mathbf{z}) : \mathbf{z} \in \mathbf{Z}_n\}$ covers the configuration space, i.e., $X \subseteq \bigcup_{\mathbf{z} \in \mathbf{Z}_n} C_n(\mathbf{z})$. Clearly, $\mathbf{Z}_n$ is a finite set, since X is compact.

Let γ_u denote the maximum distance between two points in the unit hypercube using the distance metric $\|\cdot\|$ employed in the `SetDistance` procedure. For in-

stance, if $\|\cdot\|$ is the usual Euclidean metric (L_2 norm), then $\gamma_u = \sqrt{d}$; if $\|\cdot\|$ is the L_∞ norm, then $\gamma_u = 1$.

We group the cells in $\mathbf{C}_n$ into two disjoint subsets, namely the *boundary cells* $\mathbf{B}_n$ and the *interior cells* $\mathbf{I}_n$. Let $\mathbf{B}_n'$ denote the set of all $C_n(\mathbf{z}) \subset \mathbf{C}_n$ that include a part of the obstacle set boundary, i.e., $C_n(\mathbf{z}) \cap \partial X_{\text{obs}} \neq \emptyset$. Then $\mathbf{B}_n$ contains exactly those cells that are within a cell-distance of at most $2\left(\lceil (1/\alpha)\,\gamma_u \rceil + 1\right)$ to some cell in $\mathbf{B}_n'$,

$$\mathbf{B}_n := \left\{ C_n(\mathbf{z}) : \|\mathbf{z} - \mathbf{z}'\|_\infty \leq 2\left(\lceil (1/\alpha)\,\gamma_u \rceil + 1\right) \text{ for some } \mathbf{z}' \text{ with } C_n(\mathbf{z}') \in \mathbf{B}_n' \right\},$$

where α is the constant in the constant-factor lower bound in the computation of the `SetDistance` procedure. Finally, $\mathbf{I}_n$ is defined as exactly the set of those cells that are not boundary cells, i.e., $\mathbf{I}_n := \mathbf{C}_n \setminus \mathbf{B}_n$. The reason behind our choice of the number $2(\lceil (1/\alpha)\gamma_u \rceil + 1)$ will become clear shortly.

Let $\lambda(\cdot)$ denote the Lebesgue measure in $\mathbb{R}^d$. The total number of cells, denoted by K_n, can be bounded for all n as follows [1]:

$$K_n \leq \frac{\lambda(X)}{\lambda(C_n(\mathbf{z}))} = \frac{\lambda(X)}{\left(n^{-\frac{1}{d+1}}\right)^d} = \lambda(X)\, n^{\frac{d}{d+1}} \tag{1}$$

Notice the number of cells, K_n, is an increasing and sub-linear function of n.

Let B_n denote the number of boundary cells.

Lemma 1. *There exists a constant $c_1 > 0$ such that $B_n \leq c_1 (K_n)^{1-1/d}$ for all $n \in \mathbb{N}$.*

Proof. This result follows from the fact that the obstacle boundary can be covered by $N^{(d-1)/d} = N^{1-1/d}$ cells, where N is the number of equal-size cells that cover the configuration space.

Thus the fraction of cells that are boundary cells can be bounded by

$$\frac{c_1 B_n}{K_n} = \frac{c_1 (K_n)^{1-1/d}}{K_n} = c_1 (K_n)^{-1/d} \leq c_1 \left(\lambda(X)\right)^{-1/d} n^{-\frac{1}{d+1}} = c_2\, n^{-\frac{1}{d+1}},$$

where c_2 is a constant.

We now bound the number of calls to the collision-distance procedure by examining the number of such calls separately for samples that fall into interior cells or boundary cells, respectively. Recall that C_{free} and C_{obs} are defined as the vertices that are explicitly checked and found to be in X_{free} and X_{obs}, respectively. Define $C := C_{\text{free}} \cup C_{\text{obs}}$. Our key insight is summarized in the following lemma, which will be used to derive a bound on the number of points in C that fall into interior cells.

Lemma 2. *Given any algorithm in Table 2, suppose that algorithm is run with n samples using our collision checking algorithm. Let $C_n(\mathbf{z}) \in \mathbf{I}_n$ be some interior cell. Then, there exists at most one vertex from C in $C_n(\mathbf{z})$, i.e.,*

[1] Strictly speaking, this bound should read: $K_n \leq \lceil \lambda(X)\, n^{d/(d+1)} \rceil$, where $\lceil \cdot \rceil$ is the standard ceiling function (i.e., $\lceil a \rceil$ returns the smallest integer greater than a). We will omit these technical details from now on to keep the notation simple.

$$|C_n(\mathbf{z}) \cap C| \leq 1 \text{ for all } C_n(\mathbf{z}) \in \mathbf{I}_n.$$

Before proving Lemma 2, we introduce some additional notation and establish two intermediate results. Let $\mathbf{N}_n(\mathbf{z}) \subseteq \mathbf{C}_n$ denote the set of all cells that have a cell distance of at most $\lceil \gamma_u \rceil + 1$ to $C_n(\mathbf{z})$, i.e., $\mathbf{N}_n(\mathbf{z}) := \{C_n(\mathbf{z}') : \|\mathbf{z}' - \mathbf{z}\|_\infty \leq \lceil \gamma_u \rceil + 1\}$. $\mathbf{N}_n(\mathbf{z})$ includes the cell $C_n(\mathbf{z})$ together with all its neighbors with cell distance less than $\lceil \gamma_u \rceil + 1$. The cells in $\mathbf{N}_n(\mathbf{z})$ are such that they exclude points that are sufficiently far away from points in $C_n(\mathbf{z})$ and they include points that are sufficiently far away from the boundary of the obstacle set. The last two properties are made precise in the following two lemmas.

Lemma 3. *Any point that lies in a cell outside of $\mathbf{N}_n(\mathbf{z})$ has distance at least $\gamma_u l_n$ to any point that lies in $C_n(\mathbf{z})$, i.e., $\|x - x'\| \geq \gamma_u l_n$ for all $x \in C_n(\mathbf{z})$ and all $x' \in C_n(\mathbf{z}')$ with $C_n(\mathbf{z}') \notin \mathbf{N}_n(\mathbf{z})$.*

Proof. The claim follows immediately by the construction of $\mathbf{N}_n(\mathbf{z})$.

Lemma 4. *Any point in a cell from $\mathbf{N}_n(\mathbf{z})$ has distance at least $\big(\lceil (1/\alpha)\,\gamma_u \rceil + 1\big) l_n$ to any point that lies on the obstacle set boundary, i.e., $\|x - x'\| \geq \big(\lceil (1/\alpha)\,\gamma_u \rceil + 1\big) l_n$ for all $x \in \partial X_{\text{obs}}$ and all $x' \in C_n(\mathbf{z}')$ with $C_n(\mathbf{z}') \in \mathbf{N}_n(\mathbf{z})$.*

Proof. The interior cell $C_n(\mathbf{z})$ has a cell distance of at least $2(\lceil (1/\alpha)\,\gamma_u \rceil + 1)$ to any cell that intersects with the boundary of the obstacle set. Any cell in $\mathbf{N}_n(\mathbf{z})$ has a cell distance of at most $\lceil \gamma_u \rceil + 1 \leq \lceil (1/\alpha)\,\gamma_u \rceil + 1$ to $C_n(\mathbf{z})$. Thus, by the triangle inequality (for the cell distance function), any cell in $\mathbf{N}_n(\mathbf{z})$ has a cell distance of at least $\lceil (1/\alpha)\,\gamma_u \rceil + 1$ to any cell that intersects the obstacle boundary.

Now we are ready to prove Lemma 2.

Proof (Proof of Lemma 2). The proof is by contradiction. Suppose there exists two points $x_1, x_2 \in C$ that fall into $C_n(\mathbf{z})$. Suppose x_1 is added into C before x_2. Let x_{nearest} denote the nearest point when Algorithm 1 was called with x_2.

First, we claim that x_{nearest} lies in some cell in $\mathbf{N}_n(\mathbf{z})$. Clearly, x_{nearest} is either x_1 or it is some other point that is no farther than x_1 to x_2. Note that x_1 and x_2 has distance at most $\gamma_u l_n$. By Lemma 3, any point that lies in a cell outside of $\mathbf{N}_n(\mathbf{z})$ has distance at least $\gamma_u l_n$ to x_2. Thus, x_{nearest} must lie in some cell in $\mathbf{N}_n(\mathbf{z})$. However, $\texttt{Dist}(x_{\text{nearest}}) \geq (1/\alpha)\,\|x_2 - x_{\text{nearest}}\|$ by Lemma 4; In that case, x_2 should have never been added to C, even when the `SetDistance` procedure returns α-factor of the actual distance to the obstacle set boundary. Hence, we reach a contradiction.

The following lemma provides an upper bound on the *expected* number of samples that fall into boundary cells, thus an upper bound on the expected number of points in C that fall into a boundary cell.

Lemma 5. *Let X_n denote a set of n samples drawn independently and uniformly from X. Let S_n denote the number of samples that fall into boundary cells, i.e.,*

$$S_n := \big|\{x \in X_n : x \in C_n(\mathbf{z}) \text{ with } C_n(\mathbf{z}) \in \mathbf{B}_n\}\big|.$$

Then, there exists some constant c_3, independent of n, such that $\mathbb{E}[S_n] \leq c_3\, n^{\frac{d}{d+1}}$.

Proof. Let E_i denote the event that the i-th sample falls into a boundary cell. Let Y_i denote the indicator random variable corresponding to this event. From Lemma 1 and the discussion following it, the fraction of cells that are of boundary type is $c_2 n^{-\frac{1}{d+1}}$. Thus, $\mathbb{E}[Y_i] = \mathbb{P}(E_i) = c_2\, i^{-\frac{1}{d+1}}$ and $\mathbb{E}[S_n] = \mathbb{E}\left[\sum_{i=1}^n Y_i\right] = \sum_{i=1}^n \mathbb{E}[Y_i] = \sum_{i=1}^n c_2\, i^{-\frac{1}{d+1}} \leq c_2 \int_1^n x^{-\frac{1}{d+1}} dx$, where $c_2 \int_1^n x^{-\frac{1}{d+1}} dx = \frac{c_2\,(d+1)}{d}\left(n^{\frac{d}{d+1}} - 1\right)$. Thus, $\mathbb{E}[S_n] \leq c_3\, n^{\frac{d}{d+1}}$, where c_3 is a constant that is independent of n.

The following lemma gives an upper bound on the number of points in C.

Lemma 6. *There exists a constant c_4, independent of n, such that*

$$\mathbb{E}\left[\mathrm{card}\,(C)\right] \leq c_4\, n^{\frac{d}{d+1}}.$$

Proof. On one hand, the number of points in C that fall into an interior cell is at most the total number of interior cells by Lemma 2, and thus less than the total number of cells K_n, which satisfies $K_n \leq \lambda(X)\, n^{\frac{d}{d+1}}$ (see Equation (1)). On the other hand, the expected number of points in C that fall into a boundary is no more than the expected number of samples that fall into boundary cells, which is bounded by $c_3\, n^{\frac{d}{d+1}}$, where c is a constant independent of n (see Lemma 5). Thus, we conclude that $\mathbb{E}\left[\mathrm{card}\,(C)\right] \leq c_4\, n^{\frac{d}{d+1}}$ for some constant c_4.

Finally, we are ready to prove Theorem 1.

Proof (Theorem 1). First, we show that p_{cc} is a non-increasing function of n using a standard coupling argument. We couple the events $\{I(n) = 1\}$ and $\{I(n+1) = 1\}$ with the following process. Consider the run of the algorithm with $n+1$ samples. Let A_n and A_{n+1} denote the events that the nth and the $(n+1)$st samples are explicitly checked, respectively. Clearly, $\mathbb{P}(A_{n+1}) \leq \mathbb{P}(A_n)$ in this coupled process, since the first $(n-1)$st samples are identical. Moreover, $\mathbb{P}(A_n) = \mathbb{P}(\{I(n) = 1\}) = p_{\mathrm{cc}}(n)$ and $\mathbb{P}(A_{n+1}) = \mathbb{P}(\{I(n+1) = 1\}) = p_{\mathrm{cc}}(n+1)$. Thus, $p_{\mathrm{cc}}(n+1) \leq p_{\mathrm{cc}}(n)$ for all $n \in \mathbb{N}$. Note that this implies that $\lim_{n\to\infty} p_{\mathrm{cc}}(n)$ exists.

Next, we show that $\lim_{n\to\infty}(1/n)\sum_{k=1}^n p_{\mathrm{cc}}(k) = 0$. Clearly, $\sum_{k=1}^n I_{\mathrm{point}}(k) = |C|$. Hence, $\frac{\sum_{k=1}^n \mathbb{E}[I_{\mathrm{point}}(k)]}{n} = \frac{\mathbb{E}[\sum_{k=1}^n I_{\mathrm{point}}(k)]}{n} \leq \frac{c_4\, n^{\frac{d}{d+1}}}{n} = c_4\, n^{-\frac{1}{d+1}}$, where the inequality follows from Lemma 6. Similarly, $\limsup_{k\to\infty}(\mathbb{E}[I_{\mathrm{path}}(k)] - \mathbb{E}[I_{\mathrm{point}}(k)]) = 0$, since all paths fit into an Euclidean ball with radius $o(n^{-1/(d+1)})$, which is asymptotically smaller than the side length of each cell $l_n = n^{-1/(d+1)}$. Hence,

$$\begin{aligned}\limsup_{n\to\infty} \frac{\sum_{k=1}^n p_{\mathrm{cc}}(k)}{n} &= \limsup_{n\to\infty}\left(\frac{\sum_{k=1}^n \mathbb{E}[I_{\mathrm{point}}(k)] + \mathbb{E}[I_{\mathrm{path}}(k)]}{n}\right)\\ &= \limsup_{n\to\infty} \frac{\mathbb{E}[\sum_{k=1}^n I(k)]}{n} \leq \limsup_{n\to\infty} \frac{c_4\, n^{\frac{d}{d+1}}}{n} = \limsup_{n\to\infty} c_4\, n^{-\frac{1}{d+1}} = 0,\end{aligned}$$

Finally, $p_{\mathrm{cc}}(n+1) \leq p_{\mathrm{cc}}(n)$ for all $n \in \mathbb{N}$ and $\lim_{n\to\infty}(1/n)\sum_{k=1}^n p_{\mathrm{cc}}(k) = 0$ together imply that $\lim_{n\to\infty} p_{\mathrm{cc}}(n) = 0$.

5 Experiments

To validate and demonstrate the utility of our method, we compared the performance of implementations of the RRT and RRT* algorithms, both with and without our proposed modification for collision checking. Our implementations are single threaded, utilize a kd-tree [3] for point-proximity (i.e., nearest neighbor and k- nearest neighbor) queries, and a segment-Voronoi hierarchy [2] for set distance queries. The experiments were run on a 1.73 GHz Intel Core i7 computer with 4GB of RAM, running Linux. We performed experiments on an environment consisting of a unit-square workspace with 150 randomly placed convex polygonal obstacles (see Figure 1). The goal region is a square with side length 0.1 units at the top right corner; the starting point is at the bottom left corner.

Figure 1 (Left) shows both the tree of trajectories generated by RRT*, and the set of collision-free balls that are stored in the augmented graph. After 2,000 samples, the collision-free balls have filled a significant fraction of the free space, leaving only a small amount of area uncovered near the obstacle boundaries. As proved in Section 4, any future sample is likely to land in a collision-free ball common to both it and its nearest neighbor, making future collision checks much less likely.

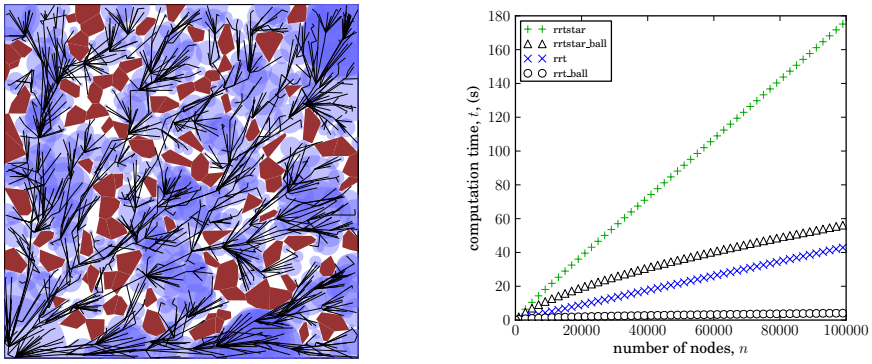

Fig. 1: (Left) Search tree, black, and balls where explicit collision checks are unnecessary, purple, for the modified RRT* algorithm with $n = 2,000$. (Right) Runtimes for the four algorithms tested, averaged over 30 runs of each algorithm, and using graph size buckets of 1000 nodes.

To generate timing profiles, both RRT and RRT* were run on the obstacle configuration in Figure 1 with and without the use of the new collision algorithm. In each run, both variants of both algorithms are run with the same initial random seed for the point sampling process so that the points sampled in each algorithm are the same (i.e., the sample sequence ω is the same for both algorithms).

Figure 1 (Right) illustrates the wall-clock measured runtime required to reach various tree sizes for each of these four implementations. Runtimes are averaged

over 30 runs and in graph-size buckets of 1000 nodes. As expected from the amortized complexity bounds, longer runs enable the proposed approach to achieve greater savings. For example, the runtime of RRT* is reduced by 40% at 10,000 vertices, and by 70% at 100,000 vertices, vs. the baseline implementation. The runtime of RRT is reduced by 70% at 10,000 vertices, and by 90% at 100,000 vertices.

The increased efficiency stemming from our proposed approach also results in an effective increase in the rate at which RRT* converges to the globally optimal solution. these implementations. Figure 2 (Left) illustrates the cost of the best path found in the baseline RRT* and the modified RRT*. In general and on average, the modified RRT* finds paths of lower cost significantly faster than the baseline RRT*.

Figure 2 (Right) illustrates the average number of explicit checks required for points which are not in collision vs. different graph sizes. As the number of samples added to the database grows, the probability of performing an explicit check decreases. At a graph size of 100,000 nodes, on average only one out of every 100 samples required an explicit check when the implementations reached that point.

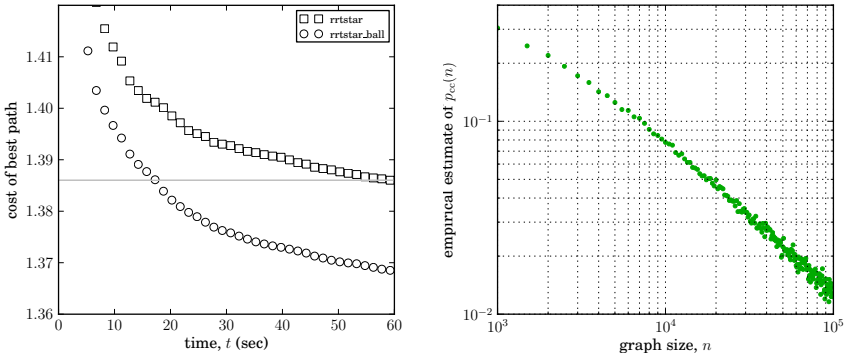

Fig. 2: (Left) RRT* best-path cost vs. time, with and without the proposed method, averaged over 30 runs and in buckets of 100 time samples. The proposed method yields significantly faster convergence. (Right) The experimental probability of performing an explicit collision query vs. graph size. Only 1% of new nodes require an explicit check when graph size is 100,000 nodes.

Figure 3 illustrates the computation of a single iteration of the four algorithms at various different graph sizes and for two different obstacle configurations. Increasing the number of obstacles increases the average iteration time while the graphs remain small; as the graph grows, the iteration times for higher obstacle count case approach those of the lower obstacle count case.

A Voronoi diagram obstacle index makes exact collision distance computation no more expensive than collision checking, however generating a Voronoi diagram in higher dimensions is prohibitively complicated. To address this, we compare the performance of the RRT with and without our proposed modification in a second

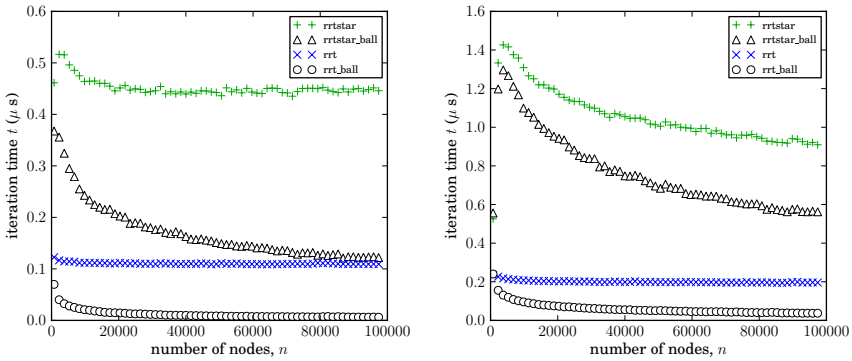

Fig. 3: The average computation time of a single iteration vs. algorithm (plot style), over 30 runs, in a configuration with 500 and 1,000 obstacles (left and right, respectively).

implementation utilizing a kd-tree for point-proximity searches, and an axis-aligned bounding box tree with Algorithm 3 for set distance queries and collision checking. Obstacles are generated as randomly placed simplices.

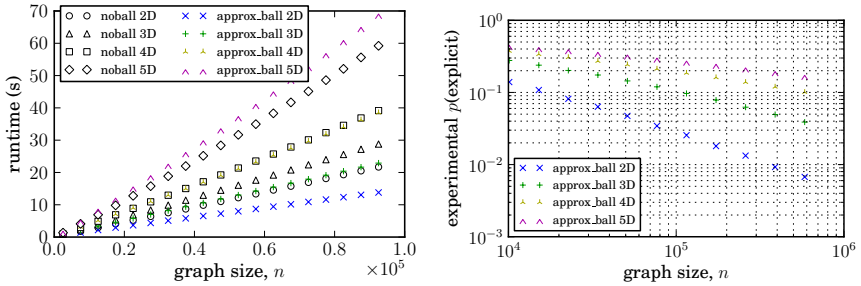

Fig. 4: Mean wall-clock runtime (Left) and experimental p_{explicit} (Right), over 20 runs, for RRT in a unit workspace $X = [0,1]^d$ for several d.

Figure 4 illustrates the measured runtime of this implementation, as well as the observed frequency of performing an explicit collision check for a unit workspace in 2 to 5 dimensions. For this choice of index, the modified algorithm using an approximate collision distance reaches a given graph size faster in 2 and 3 dimensions, at the roughly the same speed in 4 dimensions, and slower in 5 dimensions. The number of explicit collision checks is also shown. There is a significant degradation of the performance in higher dimensions. However, we note that uniform placement of obstacles is a somewhat degenerate scenario in higher dimensions. For many problems of interest, it is likely that obstacles will lie in a lower dimensional subspace,

which is a fact that the proposed algorithm is not able to exploit. Extensions to the algorithm that would be able to take advantage of lower dimensionality of the obstacle are left to future work.

6 Conclusions

In this paper, we have introduced a novel approach to collision checking in sampling-based algorithms. This approach allows us to demonstrate, both theoretically and experimentally, that collision-checking is not necessarily a computational bottleneck for sampling-based motion planning algorithms. Rather, the complexity is driven by nearest-neighbor searches within the graph constructed by the algorithms. Experiments indicate significant runtime improvement in practical implementations.

The proposed approach is very general but there are some important implementation details to be aware of when using it. First, we indicate that points which are sampled in collision are kept in order to characterize the obstacle set (i.e., in addition to those that are not in collision). While this allows that the expected number of explicit collision checks goes to zero, the number of points required to truly characterize the obstacle set can be quite large, especially in high dimension. In practice, strategies for biasing samples away from the obstacle set are likely to be more effective then keeping points which are sampled in collision. If in-collision samples are not kept and no biasing is used, the expected number of explicit checks will approach the proportion of the workspace volume which is in collision. However, even in such a case, the strategy described in this paper is effective at marginalizing the extra collision checks in the asymptotically optimal variants of sampling based motion planning algorithms. As an example, the expected runtime ratio between RRT* and RRT will be a constant which does not depend on the obstacle configuration, even if no in-collision samples are kept.

In addition, we show that calculating a sufficient approximation of the collision distance of a particular point and obstacle often does not require more computation than the worst case of performing a collision query. While this is true for a single obstacle, it is important to note that collision checking is often done using a spatial index and the choice of index may affect how the efficiency of a collision distance query compares to a simple collision query.

Also, In practice our method may benefit multi-query algorithms (e.g. PRM) and asymptotically optimal algorithms (e.g. PRM*, RRT*) more than single-query feasible planning algorithms (e.g. RRT). The latter return after finding a single path, and thus experience less ball coverage of the free space and proportionately more explicit checks, in a sparse environment. The multi-query and asymptotically optimal algorithms require more collision checks per new node, and so offer more opportunities for savings.

Lastly, some of the steps leverage the fact that in path planning problems, the straight line connecting two points is a feasible trajectory for the system. This is

in general not the case for robotic systems subject to, e.g., differential constraints. Extensions to such systems is a topic of current investigation.

Acknowledgments

This work was partially supported by the Office of Naval Research, MURI grant #N00014-09-1-1051, the Army Research Office, MURI grant #W911NF-11-1-0046, and the Air Force Office of Scientific Research, grant #FA-8650-07-2-3744.

References

1. Amato, N., Song, G.: Using Motion Planning to Study Protein Folding Pathways. Journal of Computational Biology **9**(2), 149–168 (2004)
2. Aurenhammer, F.: Voronoi Diagrams — A Survey of a Fundamental Data Structure. ACM Computing Surveys **23**(3), 345–405 (1991)
3. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Commun. ACM **18**, 509–517 (1975)
4. Bohlin, R., Kavraki, L.E.: Path Planning using Lazy PRM. In: International Conference on Robotics and Automation (2000)
5. Boor, V., Overmars, M.H., van der Stappen, A.F.: The Gaussian sampling strategy for probabilistic roadmap planners. In: IEEE Int. Conf. on Robotics and Automation, pp. 1018–1023 (1999)
6. Cortes, J., Simeon, T., Ruiz de Angulo, V., Guieysse, D., Remaud-Simeon, M., Tran, V.: A path planning approach for computing large-amplitude motions of flexible molecules. Bioinformatics **21**(1), 116–125 (2005)
7. Hsu, D., Kavraki, L.E., Latombe, J.C., Motwani, R., Sorkin, S.: On finding narrow passages with probabilistic roadmap planners. In: P. Agarwal, L.E. Kavraki, M.T. Mason (eds.) Robotics: The Algorithmic Perspective (WAFR '98), pp. 141–154. A.K. Peters/CRC Press, Wellesley, MA (1998)
8. Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. Int. Journal of Robotics Research **30**(7), 846–894 (2011)
9. Kavraki, L., Svestka, P., Latombe, J., Overmars, M.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation **12**(4), 566–580 (1996)
10. Latombe, J.: Motion Planning: A Journey of Molecules, Digital Actors, and Other Artifacts. International Journal of Robotics Research **18**(11), 1119–1128 (2007)
11. LaValle, S.: Planning Algorithms. Cambridge University Press (2006)
12. LaValle, S., Kuffner, J.J.: Randomized kinodynamic planning. International Journal of Robotics Research **20**(5), 378–400 (2001)
13. Samet, H.: Foundations of multidimensional and metric data structures. Morgan Kaufmann (2006)