Self-indexed motion planning

Article *in* Information Systems · May 2019

DOI: 10.1016/j.is.2019.04.011

Ubaldo Ruiz
Ensenada Center for Scientific Research and Higher Education
19 PUBLICATIONS **110** CITATIONS
SEE PROFILE

Edgar Chávez
Ensenada Center for Scientific Research and Higher Education
143 PUBLICATIONS **3,814** CITATIONS
SEE PROFILE

Eric S. Tellez
Consejo Nacional de Ciencia y Tecnología
70 PUBLICATIONS **426** CITATIONS
SEE PROFILE

Self-indexed Motion Planning

Angello Hoyos[a], Ubaldo Ruiz[a,c,*], Edgar Chavez[a], Eric Tellez[b,c]

[a] *Centro de Investigación Científica y Educación Superior de Ensenada, Baja California*
[b] *Centro de Investigación e Innovación en Tecnologías de la Información y Comunicación*
[c] *CONACYT Research Fellow*

Abstract

Motion planning is a central problem for robotics. A practical way to address it is building a graph-based representation (*a roadmap*) capturing the connectivity of the configuration space. The Probabilistic Road Map (PRM) is perhaps the most widely used method by the robotics community based on that idea. A key sub-problem for discovering and maintaining a collision-free path in the PRM is inserting new sample points and connecting them with the k-nearest neighbors in the previous set. Instead of following the usual solution of indexing the points and then building the PRM with successive k-NN queries, we propose an approximation of the k-Nearest Neighbors Graph using the PRM as a self-index. The motivation for this construction comes from the Approximate Proximity Graph (APG), which is an index for searching proximal objects in a metric space. Using this approach the estimation of the k-NN is improved while simultaneously reducing the total time and space needed to compute a PRM. We present simulations for high-dimensional configuration spaces with and without obstacles, showing significant improvement over the standard techniques used by the robotics community.

Keywords: Motion Planning, Approximate Proximity Graphs, Probabilistic Roadmaps.

*Corresponding author

Email addresses: `ajhoyos@cicese.edu.mx` (Angello Hoyos), `uruiz@cicese.mx` (Ubaldo Ruiz), `elchavez@cicese.mx` (Edgar Chavez), `eric.tellez@infotec.com.mx` (Eric Tellez)

1. Introduction

Motion planning is a fundamental research topic in robotics, and it has received considerable attention over the last decade [1, 2, 3]. The problem of navigating through a complex environment appears in almost all robotics applications. This problem also arises and is relevant in other domains such as computational biology, autonomous exploration, search and rescue, etc.

A central concept to the motion planning problem is the *configuration space* [1, 2, 3]. The configuration space provides a powerful abstraction that converts complicated geometric models and rigid-body transformations into the general problem of computing a path that traverses a manifold of dimension n if the robot has n degrees of freedom. Each dimension corresponds to the value of the parameter governing a joint or degree of freedom. Even if robots operate in 3D space, a single robotic arm with fingers can have, say, 40 dimensions in the configuration space.

In [4], it has been proved that even in simple cases is very expensive to compute an exact solution to the motion planning problem. For that reason, current methods are focused on finding an approximate solution to the problem. Among them, sampling-based methods are the most widely used for motion and path planning. Instead of using an explicit representation of the obstacles in the configuration space, which may result in an excessive computational cost, sampling-based methods rely on a collision detection module providing information about the feasibility of the computed trajectories.

Two favorite sampling-based techniques are Probabilistic Road Maps (PRMs) [5] and Rapidly-exploring Random Trees (RRTs) [6] which have been shown to work well in practice and possess theoretical guarantees such as probabilistic completeness[1]. One of the major drawbacks of those algorithms is that they made no claims about the optimality of the solution. Recently, Karaman et

[1]The probability of finding a feasible solution tends to one as the number of samples increases.

al. [7] proved that both algorithms are not asymptotically optimal[2]. To address that limitation, they proposed a new class of asymptotically optimal path planners named PRM* and RRT* and proved that both are probabilistically complete and asymptotically optimal.

The PRM and its asymptotically optimal variant PRM*, maintain a collision-free roadmap in the configuration space. The roadmap is a graph with nodes corresponding to collision-free sample configurations. Edges will match collision-free roads to travel between configurations. Smooth movements in the robot require dense sampling, and an increase in the degrees of freedom requires exponentially many more samples. Similarly, as the number of obstacles and the volume of their representation in the configuration space increases, the number of samples needed to improve the approximation also increases. The algorithm to build the roadmap consists of linking new inserted points with its k-nearest neighbors in the current sample.

For low-dimensional spaces and a small number of nodes in the roadmap, the k-nearest neighbor search is commonly performed using a naive brute force approach, where the distances to all nodes are computed to find the neighbors. For a large number of vertices, typically a tree-based subdivision of the configuration space is used to compute the k-nearest neighbors. In this category, the kd-tree is the most widely used, which works by creating a recursive subdivision of the configuration space into two half-spaces [8]. The kd-tree can be constructed in $O(n \log n)$ operations and the query time complexity is $O(dn^{1-\frac{1}{d}})$, where n is the number of nodes and d is the dimension of the space.

In the literature, it is well-known that the algorithms for k-nearest neighbor search degrade their performance to a sequential search as the intrinsic dimensionality of the data increases [9]. Also, most of the algorithms for k-nearest neighbor search are designed to perform fast queries, but they do not consider the time it takes to process the database and create the search data structure.

[2] When the computed solution converges to the optimal solution as the number of samples increases.

For motion-planning algorithms, the relevant measure of complexity is the total time, which includes the time spent in building the index and all the queries. The above observation poses a unique difficulty in selecting a proper index for the task because usually the cost of indexing is amortized over a large number of queries.

The key idea of our contribution is to avoid the construction of auxiliary data structures for performing the k-nearest neighbor search, as it is traditionally done. We borrow the idea of Approximate Proximity Graphs [10, 11] designed for searching in general metric spaces, and we intend to use it as a representation of the roadmap itself. Our experiments show that this approach reduces the overall time to construct the probabilistic roadmap.

We have centered our efforts on improving the efficiency of the PRM given the close resemblance between the construction of the PRM and the APG. In both cases, the graph is constructed by connecting the nodes with its k-nearest neighbors. Since the nodes are processed in random order in the APG, this favors the presence of long-length links between nodes at early stages of the process; those links are necessary to improve the precision and speed of the k-nearest neighbors search. Later on, the long-length links, which are not desirable for the PRM, are removed using a reinsertion procedure. It is relevant to mention that the APG approach cannot be directly applied to the RRT, since its construction algorithm does not favor the presence of long-length links at any stage of the process. The primary goal of the RRT is growing up a tree from an initial to a final configuration of the robot in an incremental way, while the PRM builds a roadmap of the entire configuration space that allows traveling between several configurations.

For motion-planning problems, it is critical to maximize the number of sampled configurations (nodes) per time step. Usually, a large quantity of nodes in the roadmap implies a better quality of the trajectories in the configuration space. Since these algorithms are probabilistically complete, a large number of nodes also signifies a higher probability of finding a solution [12, 13]. If the performance of the k-nearest neighbor search is improved then, more nodes can be added to the roadmap in a fixed amount of time, increasing the overall performance of the probabilistic roadmap algorithms.

2. Related work

Sampling-based motion planners use the k-nearest neighbor search heavily. The planners employ auxiliary search data structures to find and connect configurations to compute a motion plan. The literature registers several approaches to undertake the searching problem. One of the most popular is the spatial partitioning of the data. Examples of this type of algorithms are kd-trees[8], quadtrees [8] and vp-trees [14]. These data structures can efficiently handle the exact k-nearest neighbors search in lower dimensions.

Yershova and LaValle [15] proposed an extension of the kd-tree to manage $\mathbb{R}^d$ (the d-dimensional Euclidean space), S^d (the d-dimensional sphere), the group of all rotations about the origin of the three-dimensional Euclidean space $SO(3)$, and the Cartesian product of any number of these spaces. Similar to kd-trees for $\mathbb{R}^d$, their approach splits $SO(3)$ using rectilinear axis-aligned planes created by a quaternion representation of rotations. In [16], the authors report that the previous strategy performs well in many cases, but rectilinear splits produce inefficient partitions of $SO(3)$ near the corners of the partitions. They propose a method that eschews rectilinear splits in favor of splits along the rotational axes, resulting in a more uniform partition of $SO(3)$.

A strategy to improve the efficiency of kd-trees is presented in [17]. The authors describe a box-based subdivision of the space that allows focusing the search effort only in specific regions of the subdivision. In their paper, they show that the total complexity is lowered, at least in theory.

Non-Euclidean spaces, including $SO(3)$, can be searched by general nearest neighbor search data structures such a GNAT [18], cover-trees [19], and M-trees [20]. These data structures generally perform better than a sequential search. However, these methods are usually outperformed by kd-trees in practice [21].

Plaku and Kavraki [22] present a quantitative analysis of the performance of exact and approximate nearest neighbor search algorithms on increasingly high-dimensional problems in the context of sampling-based motion planning. Their analysis determines that after a critical dimension, exact nearest neighbor search algorithms examine almost all samples, becoming impractical for sampling-based algorithms when a large number of samples is required. This behavior motivates the use of approximate algorithms [21] which trade-off accuracy for efficiency.

Sandström et al. [23] explored an alternative direction for expediting the nearest neighbor search based on workspace[3] connectivity. They propose an algorithm which employs a workspace decomposition to select candidate neighbor configurations that are topologically relevant as a pre-processing step for a nearest neighbor search algorithm.

3. Background

In this section, we review the standard algorithm for constructing the PRM. We will start with some formal definitions used in the description of the algorithms.

Let $\mathcal{X}$ be the *configuration space* where $d \in \mathbb{N}$ is the dimension of the configuration space. We denote as $\mathcal{X}_{obs}$ to the *obstacle region*, and $\mathcal{X}_{free}$ as the *obstacle-free space*. The initial configuration x_{init} is an element of $\mathcal{X}_{free}$, and the goal region $\mathcal{X}_{goal}$ is a subset of $\mathcal{X}_{free}$. A *collision-free path* $\sigma : [0,1] \rightarrow \mathcal{X}_{free}$ is a continuous mapping to the free space. It is *feasible* if $\sigma(0) = x_{init}$ and $\sigma(1) \in \mathcal{X}_{goal}$. The goal of motion-planning algorithms is to compute a feasible collision-free path.

Given a graph $G = (V, E)$ where $V \subset \mathcal{X}_{free}$, a vertex $v \in G$, and $k \in \mathbb{N}$, the function Nearest_Neighbors (G, v, k) returns the k-nearest vertices in G to v.

Given two points $x_1, x_2 \in \mathcal{X}_{free}$, a boolean function Collision_Check(x_1, x_2) returns *true* if the line segment between x_1 and x_2 lies in $\mathcal{X}_{free}$ and $false$

[3]The workspace of a robot is the space in which the robot works and can be either a 3D space or a 2D surface

otherwise.

The PRM constructs a roadmap represented as a graph $G = (V, E)$ whose vertices are samples from $\mathcal{X}_{free}$ and the edges are collision-free trajectories between vertices (see Fig. 1). The PRM initializes the vertex set with n samples from $\mathcal{X}_{free}$ and attempts to connect the k-nearest points. The PRM is described in Algorithm 1. From [7], we have that for the asymptotically optimal variant PRM*, $k = e(1 + 1/d)\log(|V|)$ where d is the dimension of the configuration space, thus $k = 2e\log(|V|)$ is a good choice for all applications. Here, $|V|$ denotes the number of vertex in G.

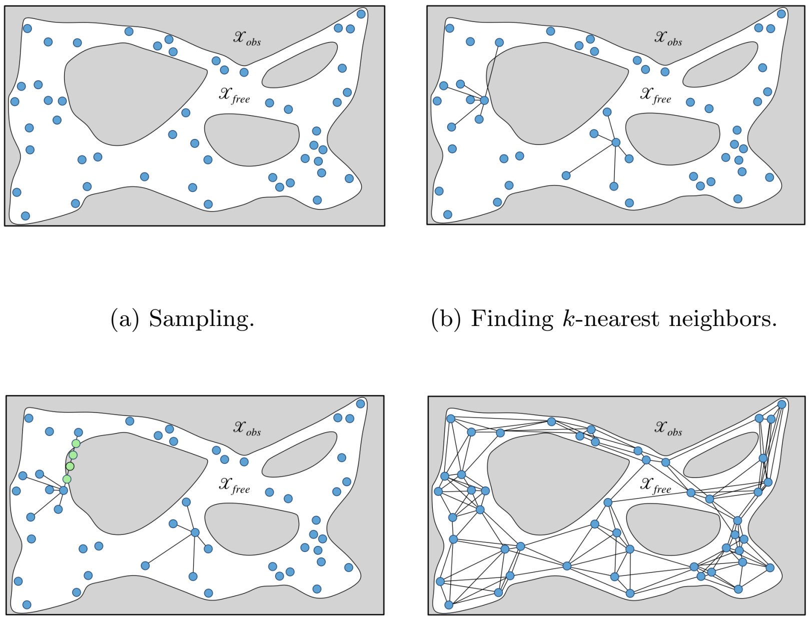
Algorithm 1 k-nearest PRM

Input: n samples from $\mathcal{X}_{free}$.

Output: A roadmap $G = (V, E)$ in $\mathcal{X}_{free}$.

```
 1: $V \leftarrow \{sample\_free_i\}_{i=1,\dots,n}$
 2: $E \leftarrow \emptyset$
 3: for each $v \in V$ do
 4:     $U \leftarrow$ Nearest_Neighbors($G, v, k$)
 5:     for each $u \in U$ do
 6:         if Collision_Check($v, u$) then
 7:             $E \leftarrow E \cup \{(v, u), (u, v)\}$
 8:         end if
 9:     end for
10: end for
11: return $G$
```

A popular variant of the PRM is the lazy-collision PRM [24, 25] or its asymptotically optimal version lazy-collision PRM*. In those variants, the Collision_Check procedure in line 6 of Algorithm 1 is omitted during the construction stage. Thus, the lazy-collision PRM or lazy-collision PRM* are built just connecting each node to its k-nearest neighbors. During the query stage, every-time a path between two vertices is searched and found in the graph; it is validated to detect if at least one of its edges is in a collision with an obstacle. If one of



(a) Sampling. (b) Finding k-nearest neighbors.

(c) Collision checking. (d) PRM

Figure 1: Construction of a Probabilistic Roadmap (PRM).

the edges in the path is in a collision, then that edge is erased from the graph, and a new path is searched again. We will use this approach of the PRM* latter in our paper.

4. Approximate proximity graph

In this section, we describe the Approximate Proximity Graph (APG), a search data structure introduced by Malkov et al. in [10, 11]. This data structure has attracted a lot of interest in the similarity search community because of its simplicity. It is constructed with successive insertions, rendering excellent searching times in high-dimensional spaces. The main idea behind the APG is to build a search graph where each node is connected to its approximate k-nearest neighbors.

A take out from studying the APG and other generalizations of the same idea is that multiple local searches give a good global approximation to the true k-nearest neighbors of the query. Local searches use two types of links, long-length and short-length. Long-length links are responsible for fast searches, while local (short-length) links are responsible for accuracy. Both types of links are a consequence of the density. Long-length links correspond to low-density sampling and appear in the early stages of the APG construction, while short-length links appear when the population of nodes is dense enough. This becomes more clear in the construction depicted in Algorithm 2 and Fig. 2. Note that this algorithm requires a definition of the function Nearest_Neighbors which computes the k-nearest neighbors in the graph. From [10], the k-nearest neighbors are found following a greedy approach using the graph itself, as described in Algorithm 3.

Algorithm 2 APG

```
Input: A set $\mathcal{U}$ of $n$ elements.
Output: A graph $G = (V, E)$ containing the $k$-nearest neighbors of each element in $\mathcal{U}$.
1: $V \leftarrow \emptyset$
2: $E \leftarrow \emptyset$
3: for each $u \in \mathcal{U}$ do
4:     $X_{near} \leftarrow$ Nearest_Neighbors$(G, u, k)$
5:     $V \leftarrow V \cup \{u\}$
6:     for each $v \in X_{near}$ do
7:         $E \leftarrow E \cup \{(u, v), (v, u)\}$
8:     end for
9: end for
10: return $G$
```

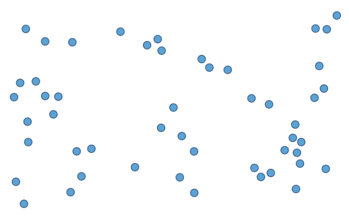
Greedy search does not guarantee to find the true k-nearest neighbor. The result depends on the initial vertex where the search started. To increase the probability of finding the proper nearest neighbor, m local searches, initiated from random vertices of the graph, can be used. This method is described in

Algorithm 3 Greedy_Search

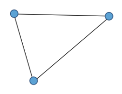
Input: A graph $G = (V, E)$, an initial vertex v_{init} and a query q.

Output: A vertex $v_{min} \in V$ whose distance to q is a local minimum.

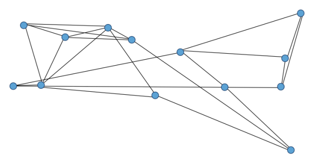
```
1:  $v_{min} \leftarrow v_{init}$
2:  $d_{min} \leftarrow$Distance($v_{min}, q$)
3:  $v_{next} \leftarrow NIL$
4:  $X_{friends} \leftarrow \{u \in V | (u, v_{min}) \in E\}$
5:  for each $u_{friend} \in X_{friends}$ do
6:      $d_{friend} \leftarrow$Distance($u_{friend}, q$)
7:      if $d_{friend} < d_{min}$ then
8:          $d_{min} \leftarrow d_{friend}$
9:          $v_{next} \leftarrow u_{friend}$
10:     end if
11: end for
12: if $v_{next}$ is not $NIL$ then
13:     return Greedy_Search($G, v_{next}, q$)
14: else
15:     return $v_{min}$
16: end if
```



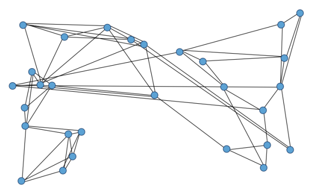
(a) Data points (the elements are processed in random order).



(b) First nodes of the graph connected with their k-nearest neighbors.



(c) Long-length links appear (a small number of nodes have been inserted in the graph).



(d) Short-length links appear (a larger number of nodes have been inserted in the graph).

Figure 2: Construction of an Approximate Proximity Graph (APG).

Algorithm 4. This algorithm requires the function Random_Vertex(G) which randomly samples a vertex from G.

Since the initial vertex is chosen at random, there is a probability p of finding the true nearest neighbor for a particular element q. This probability is non-zero because it is always possible to choose the exact nearest neighbor as the initial vertex. Let us simplify the model by assuming independent identically distributed random variables; thus if for a fixed query element q the probability of finding the true nearest neighbor in a single search attempt is p, then the probability of finding the true nearest neighbor after m attempts is $1-(1-p)^m$. Therefore, the precision of the search increases exponentially with the number of search attempts. If m is comparable to $|V|$, the algorithm becomes an exhaustive search, assuming the initial points are sampled without replacement. If G has the small-world properties [26] then it is possible to choose a vertex in a random number of steps proportional to $\log n$, maintaining an overall logarithmic search complexity.

Algorithm 4 Multi_Search

Input: A graph $G = (V, E)$, a query q, and a number m of restarts.

Output: A candidate set U of nearest neighbors of q in G.

```
1: $U \leftarrow \emptyset$
2: for $i = 1, \dots, m$ do
3:     $v_{init} \leftarrow$ Random_Vertex($G$)
4:     $v_{min} \leftarrow$ Greedy_Search($G, v_{init}, q$)
5:     if $v_{min} \notin U$ then
6:         $U \leftarrow U \cup \{v_{min}\}$
7:     end if
8: end for
9: return $U$
```

An essential parameter of the APG is the number of pseudo-nearest-neighbors connected to each newly added vertex. A large number of pseudo-nearest-neighbors increases the accuracy, while at the same time decreases the search speed. Note that this parameter also impacts the construction time of the data structure. Malkov et al. suggest $k = 3d$ where d is the dimension of the search space as a good choice for database applications where the cost to build the APG is amortized over the number of queries being solved after the construction.

In [11], Malkov et al. propose a more sophisticated algorithm to perform the search in the APG. In this algorithm, a different condition is used. The algorithm iterates on not previously visited elements close to the query, i.e., those for which the edge list has not been verified. The algorithm stops when at the next iteration the k-nearest results to the query do not change. The list of previously visited elements during the search is shared preventing repeated distance evaluations. The search algorithm is described in Algorithm 5.

5. Our approach

A natural connection can be made between the APG and the PRM. Both the APG and the PRM aim at connecting the k-nearest neighbors of the sample points (see Figs. 1d and 3). In the literature, the PRM is built using an auxiliary data structure, which needs to be constructed beforehand. In this work, we will not use an auxiliary data structure; we will instead use the same PRM for searching, adapting the APG algorithm for our purposes. Special care is needed for the PRM algorithm because two neighbors are connected if and only if a free-collision path exists between them (refer to Fig. 1). Note that in the APG there is no notion of obstacles in the space (see Fig. 3).

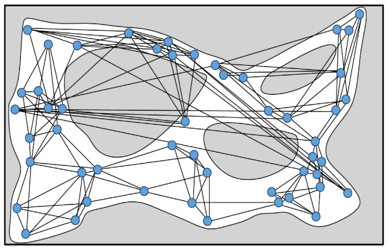


Figure 3: Standard APG constructed with the sample points in Fig. 1.

Please notice that in the configuration space, long-length edges have a higher probability of collision with obstacles. Thus most of them are not considered in the graph created by the PRM. On the other hand, long-length edges are crucial to maintaining the small-world navigation properties of the APG, in turn, responsible for the logarithmic scaling of the search. To keep the small-world properties, we propose to use the *lazy-collision* [24, 25] version of the PRM*. In this case, the Collision_Check procedure in line 6 of Algorithm 1 is

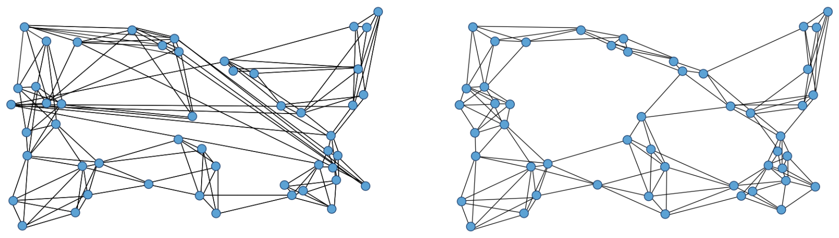
Algorithm 5 Tabu_Search

```
Input: A graph $G = (V, E)$, a query $q$, and a number $m$ of restarts.
Output: A candidate set $U$ of nearest neighbors of $q$ in $G$.
1: Let $U$ be an empty min-queue of fixed size $k$.
2: Let $C$ be an empty min-queue.
3: Let $r$ be the updated distance of the furthest element to $q$ in $U$. An empty
   $U$ defines $r = \infty$.
4: $S \leftarrow \emptyset$
5: for $i = 1, \ldots, m$ do
6:   $c \leftarrow$Random_Vertex$(V - S)$
7:   $S \leftarrow S \cup \{c\}$
8:   Append (Distance$(c, q), c)$ into $U$ and $C$
9:   loop
10:    Let $(r_b, best)$ be the nearest pair in $C$
11:    Remove $best$ from $C$
12:    if $r_b > r$ then
13:      break loop
14:    end if
15:    $X_{friends} \leftarrow \{u \in V | (u, best) \in E\}$
16:    for each $u_{friend} \in X_{friends}$ do
17:      if $u_{friend} \notin S$ then
18:        $S \leftarrow S \cup \{u_{friend}\}$
19:        Append (Distance$(q, u_{friend}), u_{friend})$ to $U$ and $C$
20:      end if
21:    end for
22:  end loop
23: end for
```

not applied during the execution, thus for a given set of vertices the constructed graph using the lazy-collision PRM* is the same as the one computed using the APG. The edges in the graph, built using the lazy-collision PRM*, are only removed once a path between two vertices in the graph is found and validated.

Another aspect to take into account is the number of nearest neighbors that are connected to each vertex. In the case of the PRM*, the value of $k = 2e\log(n)$ has been suggested to achieve good results in all applications [7]. However, the recommended parameter of the APG is $k = 3d$, with d the dimension of the search space [11]. As k increases the quality of the search improves but the time to construct the graph also increases. In our case, we are interested in maximizing the number of vertices added to the graph in a fixed amount of time, without hurting the precision of the k-nearest neighbors of each added vertex. Thus, we need to find a suitable value of k to achieve our goals.

To engage the problems described above we propose a two-step procedure. First, we build a standard APG (see Fig. 4a), which naturally contains long-length links, and then *reinsert* all nodes in the graph. This counter-intuitive step removes long-length links because when an old node is reinserted in the final graph, the density is higher and the precision of the local search increases (see Fig. 4b). After the reinsertion, it is very likely that the long-length edges will be removed and the final graph will not be as efficient as the standard APG for k-nearest neighbor searching. The lazy-collision PRM* obtained with this procedure will be closer to the true k-nearest neighbors graph. This strategy also allows to use a value of k smaller than the one suggested by Malkov et al. in [11] improving the time of construction and maintaining a good precision. In this paper, we test two approaches for reinserting the nodes. In the first one, Tabu search, described in Algorithm 5, is used to update the neighbors of each node. In the second approach, each node verifies if the second-order-neighbors are a better approximation. The details about the selection of the parameters and the experiments performed will be discussed in the next section.


(a) Standard Approximate Proximity Graph (APG).

(b) Graph after reinserting all nodes in the original APG.

Figure 4: Reinserting nodes in the Approximate Proximity Graph (APG).

6. Experiments

In this section, we present the results of using the proposed heuristics for the k-nearest neighbors search in the construction of the lazy-collision PRM*. Our results are compared with the ones obtained using three methods: 1) sequential search, 2) exact kd-trees and 3) approximate kd-trees. In the literature, we found that those methods are the standard in the robotics community to perform the search of the k-nearest neighbors in the construction of the PRM*.

To study the performance of our proposal, we define three metrics. The first one is called the *speedup*, which is the ratio between the time t_s to construct a lazy-collision PRM* using a sequential search (ground truth) for finding the k-nearest neighbors and the time t_a to construct the same data structure using an alternative algorithm for finding the k-nearest neighbors, we have

$$speedup = \frac{t_s}{t_a}$$

Fast algorithms will have a speedup bigger than one, while algorithms slower than sequential search will have a fractional speedup.

The second metric is called the *precision*, and it gives a measure of the quality of the solution obtained with the approximate search algorithms presented in this paper. Let A_i be the set of k-nearest neighbors computed by one of the alternative algorithms for a vertex i of the graph, and B_i be the corresponding set of k-nearest neighbors computed by the sequential search (ground truth). Let $|A_i \cap B_i|$ the cardinality of $A_i \cap B_i$ and $|B_i|$ the cardinality of B_i. We have that

$$precision = \frac{1}{n} \sum_{i=1}^{n} \frac{|A_i \cap B_i|}{|B_i|}$$

where n is the number of vertices in the graph. A precision value closer to one implies a better approximation of the solution.

The third metric is called the *proximity ratio*, and it is a comparison between the distances from the approximate k-nearest neighbors to a vertex i of the graph and the distances from the true k-nearest neighbors to that vertex. Let N_i be the set of k-nearest neighbors of the vertex i. The average distance from the vertex i and its k-nearest neighbors is given by

$$avg\_dist(i, N_i) = \frac{1}{k} \sum_{j=1}^{k} dist(i, j)$$

where $d(i, j)$ denotes the distance between the vertex i and the j-th element in N_i. Let A_i be the set of k-nearest neighbors computed by one of the alternative algorithms for a vertex i, and B_i be the set of k-nearest neighbors computed by the sequential search. The proximity ratio is given by

$$proximity\_ratio = \frac{1}{n} \sum_{i=1}^{n} \frac{avg\_dist(i, A_i)}{avg\_dist(i, B_i)}$$

where n is the number of vertices in the graph. Note that as the proximity ratio approaches one, the approximate k-nearest neighbors converge to the true k-nearest neighbors.

In the next experiments, we use uniform randomly generated samples from a space $\mathcal{X} = (-1, 1)^d$ where d is the dimension of the space.

We start our analysis with a characterization of the exact *kd*-tree performance in high-dimensional spaces. Fig. 5 shows the behavior of using the exact *kd*-tree in the construction of a lazy-collision PRM* in different space dimensions as the number of samples grows. In this figure, we can observe that as the space dimension increases the speedup of the lazy-collision PRM* based on an exact kd-tree decreases, becoming worse than the lazy-collision PRM* based on a sequential search. Note that in most cases, after reaching a spatial dimension of 12, the exact kd-tree starts to present an overhead compared to a sequential search due to the additional logical operations involved in the search.

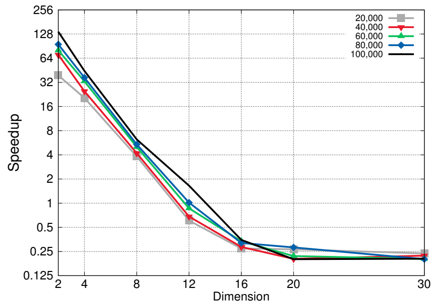
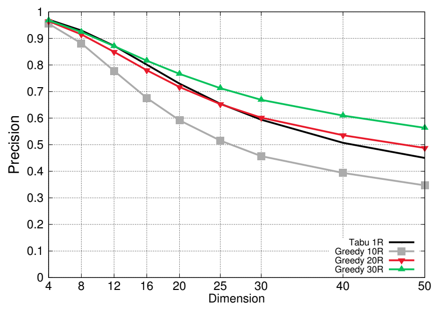
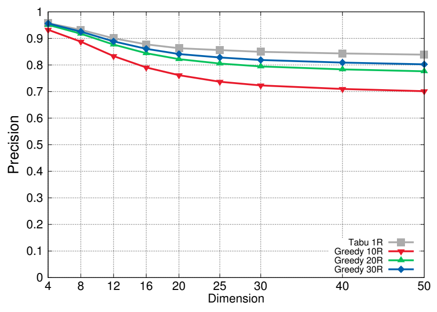


Figure 5: The speedup of constructing a lazy-collision PRM* using an exact kd-tree for the k-nearest neighbor search. Each curve corresponds to a different sample set size.

In the next experiment, we analyze the performance of Algorithms 4 (Greedy multi-search) and 5 (Tabu search) for computing the k-nearest neighbors in the construction of a lazy-collision PRM*. Following the recommendation of Malkov et al., we set $k = 3d$, where d is the space dimension. We also tested the value $k = 2e \log n$ by Karaman et al. in [7], for the construction of a PRM*. We use a set of 10000 samples in $4, 8, 12, 16, 20, 25, 30, 40$ and 50 dimensions. We set the number m of restarts to 1 for Tabu search and $10, 20$ and 30 for Greedy multi-search. Figures 6, 7 and 8 show the results of this experiment. As the number m of restarts increases in Greedy multi-search, the precision increases at the expense of the speedup. Also, it is apparent that for both algorithms, as the dimension increases, the best precision, and the best proximity ratio are obtained for $k = 3d$. However, the value of $k = 3d$ also produces the lowest speedup, in many cases worst than a sequential search, which makes it impractical for our purpose. The results also confirm that Algorithm 5 produces a better precision and a better proximity ratio in comparison to Algorithm 4. On the other hand, Algorithm 5 has the worst speedup.

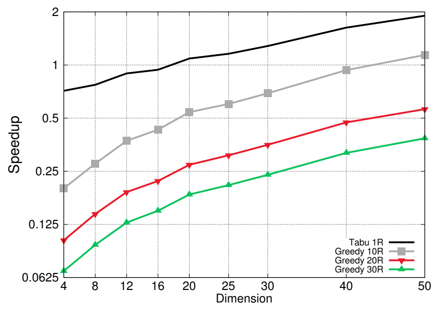


(a) $2e \log n$

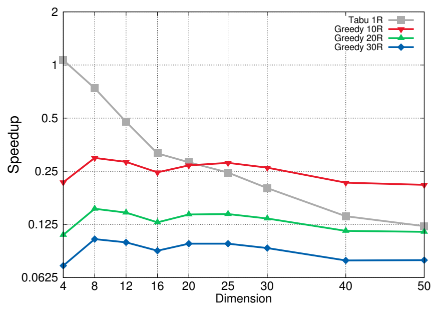


(b) $3d$

Figure 6: Precision of the approximation using the Greedy Multisearch and Tabu search.

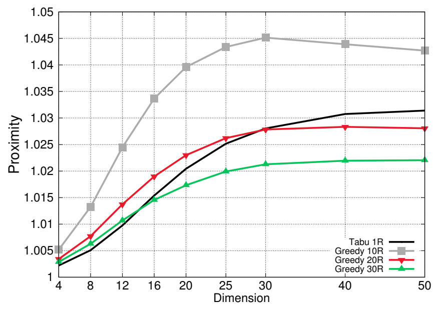


(a) $2e \log n$

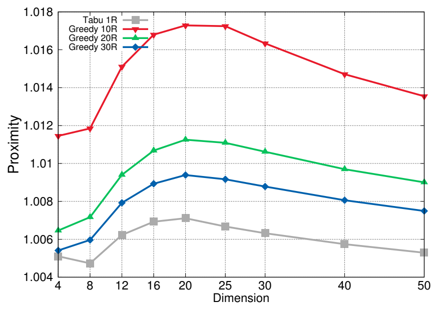


(b) $3d$

Figure 7: Speedup of the approximation using the Greedy Multisearch and Tabu search.



(a) $2e \log n$



(b) $3d$

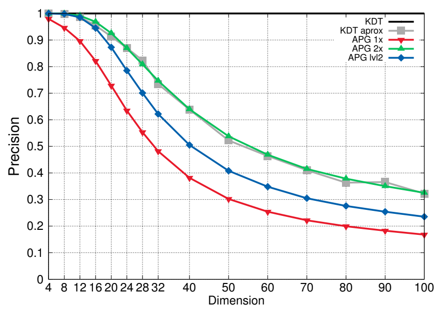
Figure 8: Proximity ratio of the approximation using the Greedy Multisearch and Tabu search.

Figures 9a, 9b, and 9c show the results of constructing an initial lazy-collision PRM* and refine the graph by reinserting all nodes updating the information about their k-nearest neighbors. We present the results of using two approaches for reinserting the nodes. In the first one, Tabu search is used to update the neighbors of each node (denoted as APG2x). In the second approach, each node verifies if the second-order-neighbors can be considered as a better approximation (denoted as APGlvl2). In this experiment, 100000 random samples were used for constructing the lazy-collision PRMs* in $4, 8, 12, 16, 20, 25, 30, 40$ and 50 dimensions. In the construction of the graph we selected $k = 2e \log n$. The results of using a standard APG with Tabu search (APG1x) and an approximate kd-tree are also included.

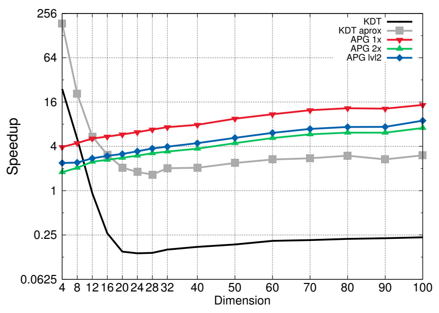
From Figs. 9a, 9b and 9c, it is possible to conclude that our algorithm APG2x has a better speedup than using a sequential search, an exact kd-tree or an approximate kd-tree achieving a similar precision for finding the k-nearest neighbors. Another interesting property is that even if the precision decreases as the dimension increases, the value of the proximity ratio in our proposal APG2x remains closer to one when compared to APG1x and APGlvl2. In other words, the approximate k-nearest neighbors computed by our algorithm are closer to the true k-nearest neighbors. We can expect that the trajectories in the lazy-collision PRM* produced with our proposal APG2x be similar to the ones obtained using an exact method for the k-nearest neighbors search like an exact kd-tree and sequential search.

In Figs. 10 and 11, we can observe that our proposal APG2x achieves similar results for the precision and proximity ratio at each dimension as the number of sample increases. Figs. 10b and 11b show an improvement of the speedup as the number of samples increases. This an expected behavior, since it takes more time to compute the k-nearest neighbors with a sequential search.

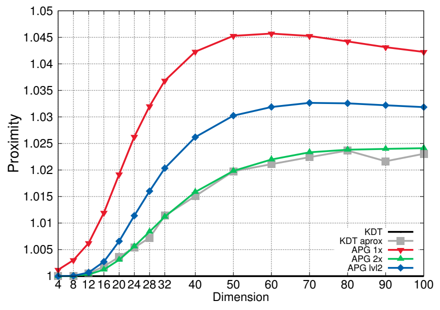
A natural concern when using an approximate algorithm for motion planning is the quality of the computed trajectories, this is analyzed in the next experiment. Here the quality measure will be the total length of the paths obtained with the lazy PRM*. In this experiment, we consider six robots moving inside



(a) Precision



(b) Speedup



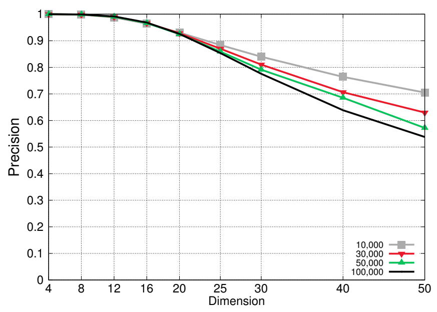
(c) Proximity

Figure 9: Precision, speedup, and proximity ratio of the two approaches for reinserting the nodes using a neighborhood $k = 2e \log n$.

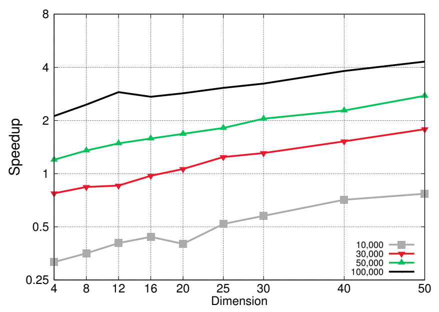
a square environment in 2D with seven obstacles (see Fig. 12). The dimension of the configuration space is 12. We sample 100000 configurations in the free space and construct five lazy PRM* using sequential scan, an approximate kd-tree, tabu search (APG1x), and the two proposed approaches (APG2x and APGlvl2) for finding the k-nearest neighbors. We perform two experiments. In the first one, we compute the trajectories ignoring the obstacles (see Fig. 13a), thus the samples are uniformly distributed. In the second one, we compute the trajectories considering the obstacles (see Fig. 13b), hence the samples are not uniformly distributed.

The results without obstacles are depicted in Fig. 13a. For each robot, our approach produces trajectories with length almost identical to the ones obtained using a probabilistic roadmap constructed using a sequential scan, the ground truth. In Fig. 13b, we can observe that when obstacles are considered in most cases our approach also produces the closest trajectories to the ones obtained performing a sequential scan during the construction.

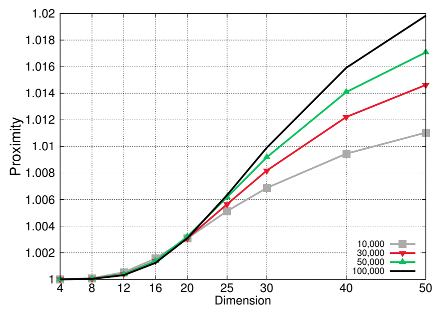
The two experimental results presented above are the outcome of a better estimation of the k-nearest neighbors with our proposed approaches.



(a) Precision



(b) Speedup

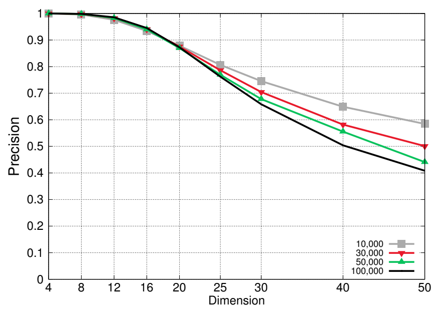


(c) Proximity

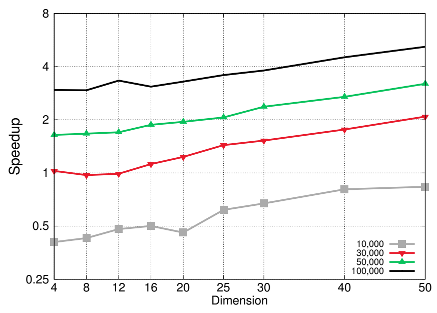
Figure 10: Precision, speedup, and proximity ratio of the approximation for different sample sizes using a neighborhood $k = 2e \log n$ and Tabu search reinserting the nodes (APG 2x).

7. Conclusions and Future work

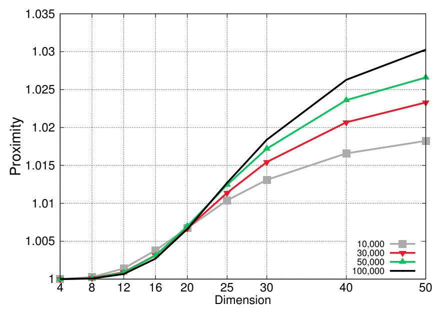
In this paper, we addressed the problem of constructing and maintaining a probabilistic roadmap in high-dimensional configuration spaces, without using an external index. We proposed a method that allows using the graph representing the roadmap as a search data structure for computing approximate k-nearest neighbors. Our approach reduces the time spent in the construction



(a) Precision



(b) Speedup

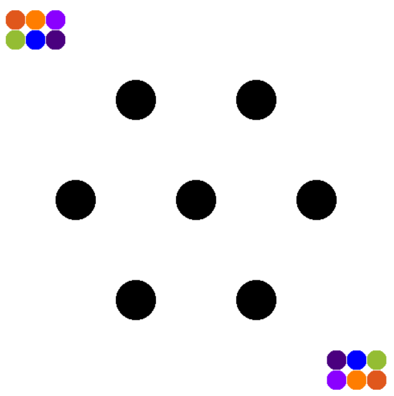


(c) Proximity

Figure 11: Precision, speedup, and proximity ratio of the approximation for different sample sizes using a neighborhood $k = 2e \log n$ and verifying for each node if the neighbors of its neighbors can be considered as a better approximation.

and saves memory since no auxiliary data structures are needed. We have shown experimentally that in high dimensional spaces, our method outperforms three methods widely used in the robotics community: 1) sequential search, 2) exact *kd*-trees and 3) approximate *kd*-trees. Our results also show that we can compute trajectories of similar quality to the ones obtained using the previous approaches in the construction of probabilistic roadmaps.

There are other applications, outside robotics, of computing a good approximation of the k-Nearest Neighbors Graph for a set of points. The algorithm can be used for example in point cloud registration, density estimation, and outlier detection. Each application poses unique challenges and should be studied in future work.


(a) Setup

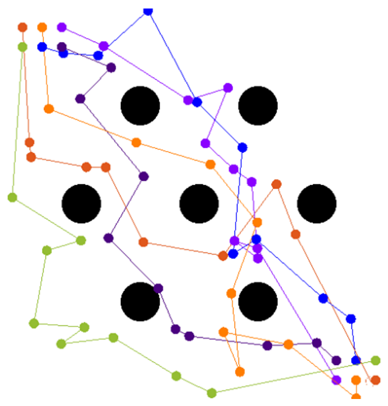
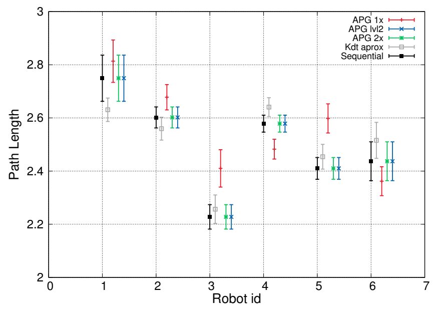
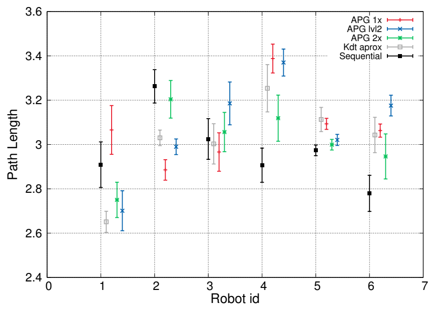

(b) Computed trajectories

Figure 12: Six robots moving inside a square environment in 2D with seven obstacles. The initial positions correspond to the color points in the upper left corner and the final positions to the color points in the bottom right corner.



(a) Free space



(b) Obstacles

Figure 13: Length of the trajectories obtained for each robot using different methods for computing the k-nearest neighbors in the construction of the probabilistic roadmap. The results for each method are grouped by robot id.

References

[1] J.-C. Latombe, Robot motion planning, Vol. 124, Springer Science & Business Media, 2012.

[2] H. M. Choset, Principles of robot motion: theory, algorithms, and implementation, MIT press, 2005.

[3] S. M. LaValle, Planning algorithms, Cambridge university press, 2006.

[4] J. Canny, The complexity of robot motion planning, MIT press, 1988.

[5] L. E. Kavraki, P. Svestka, J.-C. Latombe, M. H. Overmars, Probabilistic roadmaps for path planning in high-dimensional configuration spaces, IEEE transactions on Robotics and Automation 12 (4) (1996) 566–580.

[6] S. M. LaValle, J. J. Kuffner Jr, Randomized kinodynamic planning, The international journal of robotics research 20 (5) (2001) 378–400.

[7] S. Karaman, E. Frazzoli, Sampling-based algorithms for optimal motion planning, The international journal of robotics research 30 (7) (2011) 846–894.

[8] M. De Berg, M. Van Kreveld, M. Overmars, O. C. Schwarzkopf, Computational geometry, in: Computational geometry, Springer, 2000, pp. 1–17.

[9] E. Chávez, G. Navarro, R. Baeza-Yates, J. L. Marroquín, Searching in metric spaces, ACM computing surveys (CSUR) 33 (3) (2001) 273–321.

[10] Y. Malkov, A. Ponomarenko, A. Logvinov, V. Krylov, Scalable distributed algorithm for approximate nearest neighbor search problem in high dimensional general metric spaces, in: International Conference on Similarity Search and Applications, Springer, 2012, pp. 132–147.

[11] Y. Malkov, A. Ponomarenko, A. Logvinov, V. Krylov, Approximate nearest neighbor algorithm based on navigable small world graphs, Information Systems 45 (2014) 61–68.

[12] A. Dobson, G. V. Moustakides, K. E. Bekris, Geometric probability results for bounding path quality in sampling-based roadmaps after finite computation, in: Robotics and Automation (ICRA), 2015 IEEE International Conference on, IEEE, 2015, pp. 4180–4186.

[13] L. Janson, B. Ichter, M. Pavone, Deterministic sampling-based motion planning: Optimality, complexity, and performance, in: Robotics Research, Springer, 2018, pp. 507–525.

[14] P. N. Yianilos, Data structures and algorithms for nearest neighbor search in general metric spaces, in: SODA, 1993, pp. 311–321.

[15] A. Yershova, S. M. LaValle, Deterministic sampling methods for spheres and so (3), in: Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on, Vol. 4, IEEE, 2004, pp. 3974–3980.

[16] J. Ichnowski, R. Alterovitz, Fast nearest neighbor search in for sampling-based motion planning, in: Algorithmic Foundations of Robotics XI, Springer, 2015, pp. 197–214.

[17] M. Svenstrup, T. Bak, H. J. Andersen, Minimising computational complexity of the rrt algorithm a practical approach, in: Robotics and Automation (ICRA), 2011 IEEE International Conference on, IEEE, 2011, pp. 5602–5607.

[18] S. Brin, Near neighbor search in large metric spaces, in: 21th International Conference on Very Large Data Bases (VLDB 1995), 1995.

[19] A. Beygelzimer, S. Kakade, J. Langford, Cover trees for nearest neighbor, in: Proceedings of the 23rd international conference on Machine learning, ACM, 2006, pp. 97–104.

[20] P. M. Ciaccia, Paolo, P. Zezula, M-tree: An efficient access method for similarity search in metric spaces, in: Proceedings of the 23rd VLDB conference, ACM, 1997, pp. 426–435.

[21] A. Yershova, S. M. LaValle, Improving motion-planning algorithms by efficient nearest-neighbor searching, IEEE Transactions on Robotics 23 (1) (2007) 151–157.

[22] E. Plaku, L. E. Kavraki, Quantitative analysis of nearest-neighbors search in high-dimensional sampling-based motion planning, in: Algorithmic Foundation of Robotics VII, Springer, 2008, pp. 3–18.

[23] R. Sandström, A. Bregger, S. Thomas, N. M. Amato, Topological nearest-neighbor filtering for sampling-based planners, in: Robotics and Automation (ICRA), 2018 IEEE International Conference on, IEEE, 2018, pp. 3053–3060.

[24] R. Bohlin, L. E. Kavraki, Path planning using lazy prm, in: Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on, Vol. 1, IEEE, 2000, pp. 521–528.

[25] G. Sánchez, J.-C. Latombe, On delaying collision checking in prm planning: Application to multi-robot coordination, The International Journal of Robotics Research 21 (1) (2002) 5–26.

[26] J. Kleinberg, The small-world phenomenon: An algorithmic perspective, in: Proceedings of the thirty-second annual ACM symposium on Theory of computing, ACM, 2000, pp. 163–170.