AA274A: Principles of Robot Autonomy I
Course Notes

Oct 10, 2019

7 Motion Planning II: Sampling-based methods

Last lecture, we discussed discrete (graph-based) planning and combinatorial planning methods. Today, we will look at sampling-based planning methods for solving the motion planning problem. Our focus will be on two popular variations of sampling-based methods, PRMs and RRTs, followed by a quick survey of other algorithms, such as FMT*, and of deterministic sampling-based methods.

7.1 Introduction: From Combinatorial to Sampling-based Methods

Combinatorial methods, by definition, require exact geometric model of the robot and the obstacles. This means solutions to the combinatorial methods change as the model changes – from semi-algebraic sets to 3D triangles, from nonconvex polyhedra and others. Sampling-based methods design a motion planning algorithm independent of the collision detection module. This general philosophy has been very successful in recent years for solving problems in robotics, manufacturing, and biological applications that involve thousands or even millions of geometric primitives.

At a high level, sampling-based planning uses collision detection algorithms to probe and incrementally search the configuration space for a solution, rather than attempting to completely characterize the free space. Sampling-based algorithms are by far the most common choice for industrial-grade problems as they are conceptually simple, flexible, relatively easy to implement, and can be extended beyond the geometric case. A word of caution, however: sampling-based motion planning isn't a cure-all. Finding a solution to sampling-based methods isn't always guaranteed, and the number of samples required for convergence is often unclear. We will discuss the tradeoffs of sampling-based methods more in depth later.

7.2 Overview of sampling-based algorithms

We start with geometric motion planning.. Just as we did with the discrete (grid-based) motion planning, we will first discuss the general framework. Afterwards, we will move to more specific algorithms, mainly Probabilistic Roadmaps (PRMs) and Rapidly-exploring Random Trees (RRTs). One of the main differences between PRM and RRT methods is in its expected use: while PRM can reuse computation to solve multiple path queries using a graph, RRT is designed for single queries using a tree. The general framework we will discuss is single-query algorithm, which can easily extended to multi-query algorithms.

7.2.1 The General Framework

We first look at The algorithms follow the single-query model, which means (q_I, q_G) is given only once per robot and obstacle set. This means that there are no advantages to precomputing a graph throughout the space that could be reused, anticipating different start and goal states.

The single-query sampling-based planning algorithms are strikingly similar to the family of search algorithms summarized in Lecture 6. The main difference lies in step 3 below, in which applying an action, u, is replaced by generating a path segment, τ_s.

At a high level, the two main steps of sampling-based algorithms are (1) choosing samples in the configuration space in an unstructured way and (2) adding connections between samples. Vertices are randomly sampled in configuration space $\mathcal{C}$, and connections between the chosen vertices are examined to find valid connections. Continue sampling vertices until the algorithm finds a path from the initial vertex to the final goal, or the algorithm is terminated by a prescribed time or iteration limit. The basic idea is that once enough connections between individual samples are made, a path from initial to final state might exist. As every connection between each sample is collision free, the entire path is collision free. Points and connections add up to a final motion plan, so they all need to be valid.

Now we're ready to present these algorithms in a more rigorous manner. Most single-query, sampling-based planning algorithms follow this template:

1. *Initialization*: Let $G(V, E)$ represent an undirected search graph, for which V contains at least one vertex and E contains no edges. Typically, V contains q_I, q_G , or both. In general, other points in $\mathcal{C}_{free}$ may be included.

2. *Vertex Selection*: Choose a vertex $q_{curr} \in V$ for expansion.

3. *Local Planning*: For some $q_{new} \in \mathcal{C}_{free}$ that may or may not be represented by a vertex in V, attempt to construct a path $\tau_s : [0, 1] \longrightarrow \mathcal{C}_{free}$ such that $\tau(0) = q_{curr}$ and $\tau(1) = q_{new}$. Using this method τ_s must be checked to ensure that it does not cause a collision. If this step fails to produce a collision-free path segment, then go to step 2.

4. *Insert an Edge in the Graph*: Insert τ_s into E, as an edge from q_{cur} to q_{new}. If q_{new} is not already in V , then it is inserted.

5. *Check for a Solution*: Determine whether G encodes a solution path. As in the discrete case, if there is a single search tree, then this is trivial; otherwise, it can become complicated and expensive.

6. *Return to Step 2*: Iterate unless a solution has been found or some termination condition is satisfied, in which case the algorithm reports failure.

In the present context, each vertex in G is a configuration and each edge is a path that connects two configurations. In these notes, it will be simply referred to as a *graph* when there is no chance of confusion. Some studies refer to such a graph as a *roadmap*; however, we reserve the term roadmap for a graph that contains enough paths to make any motion planning query easily solvable.

A large family of sampling-based algorithms can be described by varying the implementations of steps 2 and 3. Implementations of the other steps may also vary, but this is less important and will be described where appropriate.

7.2.2 Probablistic Roadmap (PRM)

Previously, it was assumed that a single initial-goal pair was given to the planning algorithm. Suppose now that numerous initial-goal queries will be given to the algorithm, while keeping the robot model and obstacles fixed. This leads to a multiple-query version of the motion planning problem. In this case, it makes sense to invest substantial time to preprocess the models so that future queries can be answered efficiently. The goal is to construct a topological graph called a *roadmap*, which efficiently solves multiple initial-goal queries. Intuitively, the paths on the roadmap should be easy to reach from each of q_I and q_G , and the graph can be quickly searched for a solution.

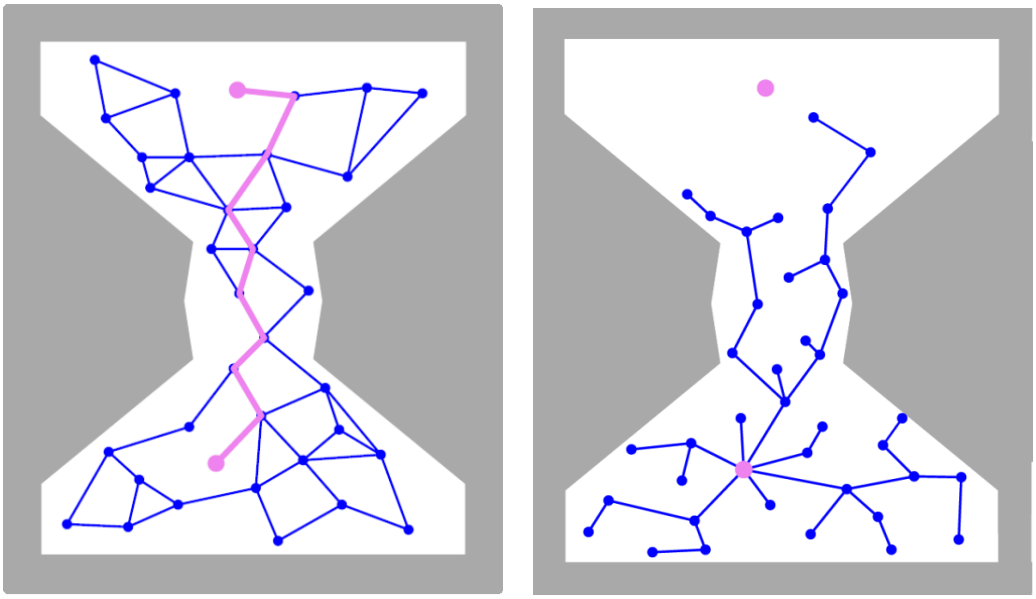
```
BUILD_ROADMAP
 1  G.init(); i ← 0;
 2  while i < N
 3      if α(i) ∈ C_free then
 4          G.add_vertex(α(i)); i ← i + 1;
 5          for each q ∈ NEIGHBORHOOD(α(i),G)
 6              if ((not G.same_component(α(i), q)) and CONNECT(α(i), q)) then
 7                  G.add_edge(α(i), q);
```

Figure 1: The basic construction algorithm for sampling-based roadmaps. Note that i is not incremented if $\alpha(i)$ is in collision. This forces i to correctly count the number of vertices in the roadmap.

The key insight of a probabilistic roadmap is to avoid the issue of creating a well characterized configuration space (which is computationally expensive), and simply randomly pick samples in the state space to analyze. At a high level, the algorithm proceeds as follows:

First, throw a bunch of random samples into your state space, discarding forbidden configurations. Then pairs of nearby samples (a pair of $x, x' \in \mathcal{X}_n s.t ||x - x'|| \leq r$, where r is a prescribed distance threshold) are connected. Next, a collision checker – as computationally inexpensive as possible – tests whether the transition between any two samples in the state space results in a constraint violation. Figure 1 illustrates this process in more rigorous manner.

After you iterate through all the samples and add all the relevant edges, you should have a graph that extends from x_{init} to x_{goal} (Figure 2a). You can then use a shortest-path algorithm like A^* to compute the optimal path from the starting pose to the goal pose.



(a) Example solution found via PRM (b) Example exploration tree by RRT.

Often we would like to connect all the random samples in an efficient manner. A relatively intuitive technique to doing so is to simply place an arbitrary disk (called the disk connectivity radius, Figure 3) around each sample x_i and add an edge between x_i and all samples that are within the disk and are collision-free.

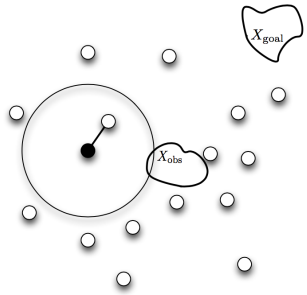


Figure 3: PRM Collision Checking

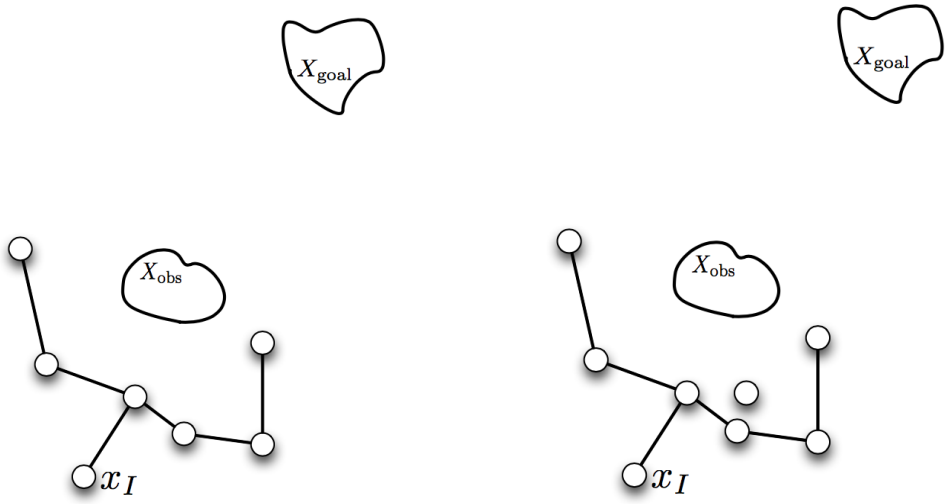
There is a caveat to PRM: while it tends to provide a good characterization of the state space, you need to perform many costly collision checks. During each check the PRM algorithm attempts to connect a new node in the state space with all potential neighbors, which can add up to a great deal of computation.

7.2.3 Rapidly-exploring Random Trees (RRT)

RRT is one of the most popular variations of an incremental sampling and searching approach that yields good performance in practice without any parameter tuning. The idea is to incrementally construct a search tree that gradually improves exploration resolution but does not need to explicitly set any resolution parameters. In the limit, the tree densely covers the space. A dense sequence of samples is used as a guide in the incremental construction of the tree. If this sequence is random, the resulting tree is called a rapidly exploring random tree (RRT). In general, this family of trees, whether the sequence is random or deterministic, will be referred to as rapidly exploring dense trees (RDTs) to indicate that a dense covering of the space is obtained.

The Rapidly-exploring Random Trees (RRT) algorithm was developed to respond to the issues with Probabilistic Roadmaps (PRM). As discussed, PRMs produce high quality motion plans but use a very dense characterization of the state space to do so. RRT on the other hand, is a lightweight approach. Rather than attempting to characterize the entire state space we attempt to build a "portfolio"of trajectories incrementally as the algorithm progresses.

Suppose you have already somehow developed two trajectories as seen in Figure 4a.



(a) Initial "portfolio" of trajectories

(b) RRT New Sample

Notice that this portfolio is incomplete - we are trying to expand it to get to our goal region. At each step of the algorithm, we randomly generate a new sample in the state space, Figure 4b.

The newly generated sample now must be connected to our tree. Different variations of RRT distinguish themselves with a wide range of connection strategy, but the simplest way is to connect the new node to the closest node already on the graph. We keep repeating this process until we reach the goal region. Without any optimization, the quality of paths that RRT produces can be poor. RRT* that includes an optimization scheme to improve upon RRT.

7.2.4 Performance of RRT

RRT, especially its asymptotically optimal variation RRT*[1], is known to perform very well in practice, and its performance can be attributed to Voronoi bias.

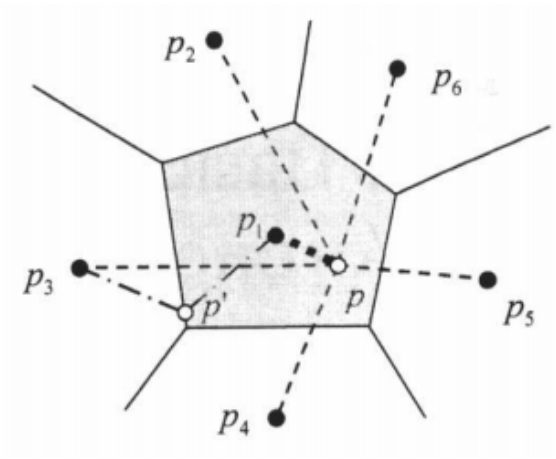


Figure 5: A basic Voronoi diagram. [Dob05]

Before we see how Voronoi bias improved RRT, we define Voronoi tessellation. The Voronoi diagram of a set of "sites" or "generators" is a collection of regions that divide up the plane. Each region corresponds to one of the sites or generators, and all of the points in one region are closer to the corresponding site than to any other site. Where there is not one closest point, there is a boundary. Note that in Figure 5, the point p is closer to p_1 than to any other enumerated points. Also note that p_0, which is on the boundary between p_1 and p_3, is equidistant from both of those points [Dob05]. Voronoi bias refers to geometric properties that come from sampling based on Voronoi tessellation of the configuration space.

RRT finds optimal solutions as it explores with Voronoi bias. Recall RRTs are incremental search algorithms; they incrementally construct a tree from the initial state to the goal state (bidirectional versions exist as well). At each step, a random sample is taken and its nearest

[1]In RRT*, we can first try to optimize the way we connect a random new node x_i to our graph. After we find the node x_{near} in the graph, we can analyze the neighbors of x_{near} to see if there is a cheaper way to connect x_i to the graph. This method introduces a notion of local optimality to the algorithm. We can also check if it is possible to get to any of the neighbors of x_near faster if we were to go through x_i. Again, remember we must be using the collision checker each time we add an edge to the graph.

neighbor in the search tree computed. A new node is then created by extending the nearest neighbor toward the random sample. This exploration strategy results in discretization of space in Voronoi diagram: at each iteration, the probability that a node is selected is proportional to the volume of its Voronoi region; hence, search is biased toward those nodes with the largest Voronoi regions (representing unexplored regions of the configuration space). This causes RRTs to rapidly explore. Alternatively, RRTs can be seen as attempting to decrease dispersion. The random sample then becomes an estimate of the center of the largest empty ball, and its distance to its nearest neighbor an estimate of the dispersion. The RRT then grows toward that sample, attempting to decrease the dispersion.

7.2.5 Theoretical evaluation of sampling-based methods

One of the main challenges of sampling-based motion planning is that it is often unclear how many samples we need to guarantee convergence on our solution. Both PRM[2] and RRT were proven to eventually converge to a solution as the number of samples, $n \to \infty$, if the solution exists [LaV98, KSO94]. [KF11] shows, however, that RRT can produce arbitrarily-bad paths with non-negligible probability. Sampling-based motion planning is still an active area of research, and we will leave rigorous evaluation of sampling-based methods to readers. However, we can offer a general summary based in empirical results:

- PRM-like motion planning algorithms tend to find a good (not necessary optimal) path for a given number of nodes, n. However, PRM's biggest tradeoff is the costly collision checks.
- RRT-like motion planning algorithms tend to find a feasible path very quickly. However, RRT-based path tends to be of poor quality. Without the asymptotically optimal flavor, RRT tends to be trapped by disallowing new better paths to emerge.

7.3 Fast Marching Tree Algorithm (FMT*)

Ideally, we want to combine the features of both single-query algorithms (chiefly RRT, which is quick but poor) and multiple-query algorithms (chiefly PRM, which requires a large number of costly collision checks). One way is to run a dynamic programming algorithm called the Fast Marching Tree Algorithm (FMT*) on sample nodes in a way that allow us to grow the tree in cost-to-arrive space. This is considered a "lazy" method, reducing the number of collision checks while still maintaining guarantees of asymptotic optimality.

[2]For constant r.

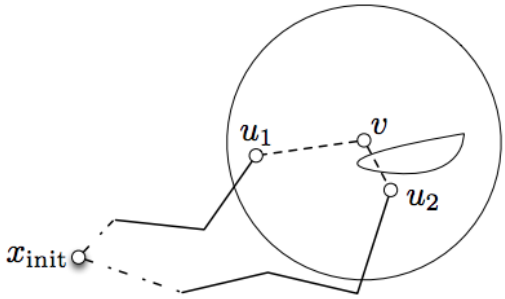


Figure 6: Idea of FMT*

In FMT*, similar to PRM, we first sample a set of nodes to use for our planning. However, unlike PRM, we incrementally perform collision-checking on edges in order to form a tree of collision-free paths for each query. In particular, the process for adding a new node is as follows. Suppose we are extending an existing tree as shown in Figure 6. Given the lowest-cost open node in the tree, we find its unvisited neighbors in the sample set, and designate those as the next candidate samples for connection to the tree. For each of those candidate samples, we find the neighboring open node in the tree which will provide the lowest cost-to-arrive to the candidate sample, *lazily* ignoring the presence of obstacles. Next, we check whether actually establishing that connection would be collision-free, and if it is, we add that connection to the tree. Thus FMT* makes use of the lazy dynamic programming operator:

$$c(v) = \min_{u:\|u-v\|<r_n} \text{Cost}(u, v) + c(u) \tag{1}$$

where u are nodes within radius r_n of candidate connection node v, $\text{Cost}(u, v)$ is the cost of an edge between u and v, and $c(u)$ is the cost-to-arrive of u.

However, crucially, if a connection attempted in this way is not collision-free, we do not attempt to connect the candidate sample to other open neighbors in the tree. This is part of the lazy collision-checking that is a key feature of FMT*. It is not perfect, as we sometimes mistakenly choose the neighbor that has a collision, ultimately resulting in some suboptimal connections. However, sophisticated analysis shows the number of times that getting the neighbor wrong goes asymptotically to zero as the number of points goes to infinity, which means this algorithm is able to recover an optimal solution even though it only performs one collision check for each attempted connection. You can show that the solution quality that you get is very close to PRM with a computational complexity that is very close to RRT thanks to the laziness of FMT*.

The pseudocode of FMT* is shown in Figure 7:

Fast Marching Tree Algorithm (FMT*): Basics

Require: Sample set V comprised of x_{init} and n samples in $\mathcal{X}_{\text{free}}$, at least one of which is also in $\mathcal{X}_{\text{goal}}$

1: Place x_{init} in V_{open} and all other samples in $V_{\text{unvisited}}$; initialize tree with root node x_{init}
2: Find lowest-cost node z in V_{open}
3: For each of z's neighbors x in $V_{\text{unvisited}}$:
4: Find neighbor nodes y in V_{open}
5: Find locally-optimal one-step connection to x from among nodes y
6: If that connection is collision-free, add edge to tree of paths
7: Remove successfully connected nodes x from $V_{\text{unvisited}}$ and add them to V_{open}
8: Remove z from V_{open} and add it to V_{closed}
9: Repeat until either:
1 V_{open} is empty $\Rightarrow$ report failure
2 Lowest-cost node z in V_{open} is in $\mathcal{X}_{\text{goal}}$ $\Rightarrow$ return unique path to z and report success

Figure 7: Pseudocode for FMT*

8 Kinodynamic Planning

The basic geometric case, where a robot does not have any constraints on its motion and only an collision-free solution is required, is well-understood and solved for a large number of practical scenarios. However, robots do usually have stringent kinematic/dynamical constraints on their motion, which in most settings need to be properly accounted for.

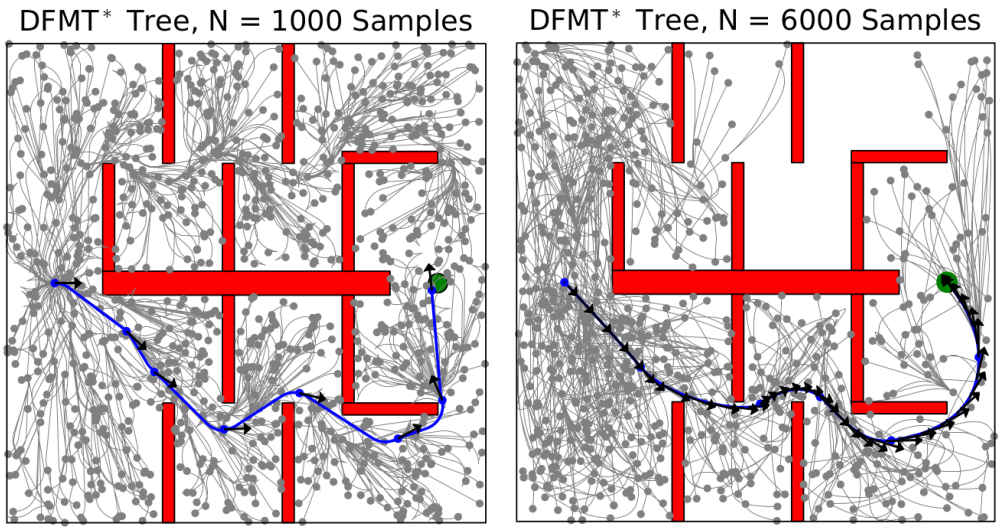
Kinodynamic motion planning problems are those where feasible paths are subject to differential constraints in addition to obstacle avoidance. There are two common types of system dynamics that will be at play in kinodynamic planning: the driftless case and the drift case. *Driftless control-affine systems* have well-understood conditions for small time local controllability and established methods for local steering. Trajectories x in the configuration space must satisfy $\dot{x} = \sum_{i=1}^{m} g_i(x)u_u$. *Control-affine systems* with drift are difficult in general to guarantee local controllability for, and local steering is known only for special cases. The dynamics can be described as $\dot{x} = Ax + Bu + c$.

1: $\mathcal{T}$.init(x_{init})
2: **for** $i = 1$ to k **do**
3: $x_{\text{rand}} \leftarrow$ RANDOM_STATE()
4: $x_{\text{near}} \leftarrow$ NEAREST_NEIGHBOR($x_{\text{rand}}, \mathcal{T}$)
5: $x_{\text{new}} \leftarrow$ NEW_STATE($x_{\text{rand}}, x_{\text{near}}, \eta$)
6: **if** COLLISION_FREE($x_{\text{near}}, x_{\text{new}}$) **then**
7: $\mathcal{T}$.add_vertex(x_{new})
8: $\mathcal{T}$.add_edge($x_{\text{near}}, x_{\text{new}}$)
9: **return** $\mathcal{T}$

Figure 8: Generic algorithm for geometric-RRT

The previously described RRT can be extended to the kindodynamic case with relative simplicity. Figure 8 provides such implementation. While this algorithm is similar to RRT in that it draws a random state and finds its nearest neighbor x_{near}, the algorithm samples a random control $u \in U$ and random duration, which are forward propagated for time t from x_{near} to find a kinodynamic solution.

Figure 9 shows the trajectories planned by Differential FMT* (DFMT*) for the Reeds-Shepp car (driftless case) and double integrator system (drift case). Note how the trajectories differ between the two systems.



(a) Reeds-Shepp car (b) Double integrator system

Figure 9: DFMT* Tree in driftless case (left) and drift case (right)

9 Deterministic Sampling-Based Motion Planning

Probabilistic sampling-based algorithms, such as the probabilistic roadmap (PRM) and the rapidly exploring random tree (RRT) algorithms, represent some of the most successful approaches to robotic motion planning, due to their strong theoretical properties (in terms of probabilistic completeness or even asymptotic optimality) and remarkable practical performance. Such algorithms are probabilistic in that they compute a path by connecting independently and identically distributed (i.i.d.) random points in the configuration space. This randomization aspect, however, makes several tasks challenging, including certification for safety-critical applications and use of offline computation to improve real-time execution. Hence, an important open question is whether similar (or better) theoretical guarantees and practical performance could be obtained by considering deterministic, as opposed to random sampling sequences.

In order to answer this question, we first provide a review of low-dispersion sampling with a focus on l_2-dispersion. For a finite set S of points contained in $\chi \in R^d$, l_2-dispersion $D(S)$ is defined as:

$$D(S) := \sup_{x \in \chi} \min_{s \in S} ||s - x||_2 \tag{2}$$

Intuitively, l_2-dispersion quantifies how well a space is covered by a set of points S in terms of the largest Euclidean ball that touches none of the points. A smaller ball radius would mean that the points are more uniformly distributed. It should be noted that deterministic sequences exist with $D(S)$ of order $O(n-1/d)$ (d denotes the number of dimensions), referred to as low-dispersion sequences. However, such a sequence minimizing l_2-dispersion is only known for $d = 2$.

As it turns out, optimality can indeed be recovered wen using deterministic sampling. In particular, let c_n denote the arc length of a path returned with n samples. Then if the sample set S has dispersion $D(S) \leq \gamma n^{-1/d}$ for some $\gamma > 0$ and $\lim_{n\to\infty} n^{1/d} r_n = \infty$, then $\lim_{n\to\infty} c_n = c^*$, where c^* is the cost of an optimal path.

References

[Dob05] Adam Dobrin. A review of properties and variations of voronoi diagrams. *Whitman College*, pages 1949–3053, 2005.

[KF11] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *CoRR*, abs/1105.1186, 2011.

[KSO94] Lydia Kavraki, Petr Svestka, and Mark H Overmars. *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*, volume 1994. Unknown Publisher, 1994.

[LaV98] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.