CS 497 SML Presentation

Shai Sachs

October 11, 2001

In this discussion we focus on two papers: *Quasi-Randomized Path Planning* [QRPP01] and *Path Planning in Practice: Lazy Evaluation on a Multi-resolution Grid* [PPP01]. Both of these papers approach the path-planning problem by deterministically searching through a discrete set of points, considering a variety of optimizations to earlier grid-based approaches.

1 Evaluation

1.1 Point sets

Although most algorithms are judged by their efficiency and efficacy, [QRPP01] asserts that discretized path-planning algorithms should be judged at a finer resolution, by evaluating the quality of the discrete point sets on which they rely. The paper suggests two ways to evaluate discrete point sets.

- Discrepancy

  Discrepancy is loosely defined as "the degree of difference of a set of points (measured by some criteria) with respect to the properties of points that are uniformly distributed in space." In order to measure discrepancy, the properties of uniformly-distributed points and the criteria by which differences are measured must be fixed. In [QRPP01], the authors consider the distribution of points with respect to axis-aligned rectangles. The discrepancy of a point set P with respect to an axis-aligned rectangle R is measured as

  $$D(P, R) = n \cdot vol(R) - |P \cap R|$$

  In particular, the authors define the L_∞-discrepancy of the first N points of a d-dimensional sequence, $\mathbf{S}$ in $[0,1]^d$ as:

  $$D_N(P) = \sup_{[\mathbf{a}, \mathbf{b}) \subset [0,1]^d} \left| \frac{|\{\mathbf{s}_1, \ldots, \mathbf{s}_N\} \cap [\mathbf{a}, \mathbf{b})|}{N} - \prod_{i=1}^{d} (b_i - a_i) \right|$$

  The supremum, denoted sup, is the lowest upper bound on a set of values.

We might think of discrepancy as the "sparseness" of a point set. High discrepancy (as measured above) suggests that some rectangles in the region $[0,1]^d$ are covered by a small number of sample points.

- Dispersion

  The dispersion of a point set is loosely defined as "the furthest distance between any point in the space and its nearest sample point." The authors suggest visualizing dispersion by placing a ball with radius equal to the dispersion at each sample point. If the dispersion is small, than a query can be easily connected to the sample points; also, the sample points are thought to cover the space densely, in some sense.

  Although no exact way to measure dispersion is given (aside from the idea of using Monte-Carlo methods), the dispersion $d(P)$ of a set of points is related to its discrepancy $D(P)$ by

$$d(P) \leq (D(P))^{\frac{1}{d}}$$

1.2 The Algorithms

- Randomized vs. Deterministic – Since a randomized algorithm rarely behaves identically on repeated trials, it is hard to compare a randomized algorithm with a deterministic one. In [QRPP01], this difficulty is met by comparing the performance of the deterministic algorithm with the "average" performance of 20 iterations of the randomized algorithm.

- Size of Solution Path – [QRPP01] measures the success of an algorithm by the size of its solution path, hypothesizing that a smaller solution path yields to a lower running time.

- Computation Time – [QRPP01] also measures the running time of algorithms, which is used to compare Lazy-PRM and Quasi-random Lazy-PRM algorithms.

- Number of Collision Checks – The work in [PPP01] is motivated by the observation that collision checking is expensive; therefore, algorithms are evaluated by the number of collision checks required.

2 Path-Planning Algorithms

2.1 Probabilistic Roadmap

This algorithm is described simply in [QRPP01] as follows: Repeatedly pick a random node n in C_{free}. Find the possible neighbors of that node. If an obstacle-free path between n and its neighbor n', add it to the set of possible paths. Following this "roadmap construction" phase, the algorithm searches through the roadmap for a path from the initial to the goal state (using, for example, the A* graph search algorithm.)

2.2 Quasi-Probabilistic Roadmap

This algorithm uses the same strategy as that in section 2.1, except that instead of randomly choosing a node, it deterministically chooses one. [QRPP01] proposes two ways of deterministically choosing nodes.

Using the *Hammersley point set*, the i^{th} point is defined as

$$\left(\frac{i}{N}, r_{p_1}(i), \ldots, r_{p_{d-1}}(i)\right)$$

Using the *Halton point set*, the i^{th} point is defined as

$$(r_{p_1}(i), \ldots, r_{p_d}(i))$$

Where p_j is one of $d-1$ distinct primes (the authors suggest the first $d-1$ primes) and $r_p(i)$ is the p-ary notation for i written in reverse order. In both cases, $0 \leq i \leq N-1$.

These point sets were chosen because they have low dispersion and low discrepancy.

2.3 Lazy-PRM

This algorithm provides a slight variation to that presented in 2.1 by neglecting to check for collision as the roadmap is constructed. If, during the roadmap search, a node is found to be in collision, the node is deleted from the roadmap, and the search is repeated.

2.4 Quasi-random Lazy-PRM

This algorithm further modifies the algorithm of section 2.3, resulting in a "lazier" method, so named because the points are never explicitly represented. Instead, a lattice is implicitly defined by a set of N points, in which the i^{th} point is $\frac{i\mathbf{z}}{N}$. $\mathbf{z}$ is a carefully chosen integer vector so that the resulting point set has low discrepancy. Because the points of a lattice form an Abelian group, it is possible to determine all of the nearest neighbors of any point v in time $O(d)$, where d is the dimension of the lattice: simply add or subtract each of the d generators of the group from v.

Note that it is also easy to increase the "resolution" of the lattice. Because no points are stored, the algorithm can use a finer-grain equation for the neighbors of v and the search automatically proceeds over a denser lattice.

[QRPP01] and [PPP01] share the main ideas used in this algorithm. Both suggest increasing the resolution of the roadmap when a search fails, and both neglect to explicitly store all the points in the roadmap.

2.5 Lazy Evaluation on a Multi-Resolution Grid

This work was motivated by the observation that collision detection is expensive - for the model the author used, each call to the collision checker took about

0.08 seconds. As stated above, the work focuses on two key ideas. First, the grid need not be explicitly represented; second, searching the grid at different "resolutions" may save considerable time.

The grid is represented as the intersection of a series of hyperplanes. The global planner "enables" certain hyperplanes, allowing the local planner to search through the set of points in the intersection of enabled hyperplanes, using a graph search algorithm such as A*. While it searches, the local planner checks for collision along the candidate path using a scheme which minimizes the number of collision checks needed to disqualify the path, while maximizing the number of other colliding paths that are also disqualified. If the local planner fails, the global planner enables a few more planes and restarts the A* search.

3 Discussion Questions

1. **The Value of Randomization** – Randomization can be a useful tool in escaping local minima, and it can be a reasonable way of sampling a set whose properties we have prior knowledge of (for example, finding witnesses of compositeness in the randomized primality test.) When constructing a dense roadmap, we seek to densely cover the entire space. Do either of the common reasons for applying randomization apply to roadmap methods?

2. **Randomness vs. Pseudo-Randomness** – [QRPP01] makes the claim that "any machine implementation generates a deterministic sequence of pseudo-random numbers," and that therefore randomized path planning cannot beat an "adversary" who tries to design a hostile configuration space. On many machines, where the pseudo-random seed is chosen from a small space, or is not chosen from a uniform distribution, this statement might be true. However, is it possible to design a pseudo-random number generator which, faced with a sufficiently weak adversary, makes the numbers "look" random, perhaps by choosing the seed from a large space?

3. When considering randomized path planners, it is common to think of an "adversary" who chooses some configuration space. A superior path planner can efficiently find a path in the space, even in the face of a powerful adversary. Does it make sense to apply it to path planning?

4. Is it fair to compare the performance of deterministic and randomized algorithms?

5. Lydia Kavraki, who won the Grace Murray Hopper Award for her work on PRM, writes that "PRM particularly suitable for many-dof robots performing several point-to-point motions in known static workspaces." Does this description apply to the problems considered in [QRPP01]? If not, would PRM perform better against Q-PRM in problems that do meet this description?

6. [QRPP01] makes a number of interesting theoretical contributions, and it provides some proof of its efficiency – but only in one configuration space. Can we imagine other configuration spaces in which this planner compares poorly to a probabilistic roadmap method?

7. The algorithm in [PPP01] restarts A* when the local planner fails. This step might be the source of considerable inefficiency: each call to A* might take considerable time, because the number of points in a set of enabled hyperplanes (though a small fraction of the overall number of points) might still be quite large. Under what conditions do we expect the local planner to succeed at coarser resolutions, thereby avoiding the expensive step of restarting A*? Or do we believe that restarting A* might not be as expensive as suggested above?

8. Consider the test cases provided in [PPP01] (that is, Configurations A – J in Figures 2 and 3). What does the configuration space look like for this problem? What does the obstacle space look like? Is it possible that some planner could solve this problem much more quickly?

9. What do we think about the ideas shared by [QRPP01] and [PPP01] – representing the grid implicitly and searching over a variety of different resolutions? Which paper seems to implement these ideas more efficiently? Could the other paper's implementation be improved?

10. [QRPP01] suggests evaluating the roadmap itself, as well as the overall running time of the algorithm which uses it. Is this idea productive? Can we imagine other methods of measuring roadmaps besides dispersion and discrepancy, which might be more fruitful?