

An Adaptive Framework for ‘Single Shot’ Motion Planning ^{*}

Daniel R. Vallejo

Christopher Jones

Nancy M. Amato

Texas A&M University
dvallejo@cs.tamu.edu

Sandia National Laboratories
cvjones@sandia.gov

Texas A&M University
amato@cs.tamu.edu

Abstract

This paper proposes an adaptive framework for single shot motion planning, i.e., planning without preprocessing. This framework can be used in any situation, and in particular, is suitable for crowded environments in which the robot’s free C-space has narrow corridors such as maintainability studies in complex 3D CAD models. Our iterative strategy adaptively selects a planner whose strengths match the current situation, and then, on-line, switches to a different planner when circumstances change. This requires techniques to evaluate the characteristics of the current query, and a set of planners which are characterized so that we can match the query with the best planner for it. Our experimental results in complex 3D CAD environments show that our strategy solves queries that none of the planners could solve on their own.

1 Introduction

Automatic motion planning has application in many areas such as robotics, virtual reality systems, and computer-aided design. Although many different motion planning methods have been proposed, most are not used in practice since they are computationally infeasible except for some restricted cases, e.g., when the robot has very few degrees of freedom (dof) [13]. Indeed, there is strong evidence that any complete planner (one that is guaranteed to find a solution or determine that none exists) requires time exponential in the number of dof of the robot. For this reason, attention has focussed on randomized or probabilistic motion planning methods.

When many motion planning queries will be

performed in the same environment, then it may be useful to preprocess the environment with the goal of decreasing the difficulty of the subsequent queries. Examples are the *roadmap* motion planning methods [13], in which a graph encoding representative feasible paths is built, usually in the robot’s configuration space (the parametric space representing all possible placements of the robot in the workspace). Queries are processed by connecting the initial and goal configurations to the roadmap, and then finding a path in the roadmap between these two connection points. Recently, randomized or *probabilistic roadmap methods* (PRMs) have gained much attention for problems involving high-dimensional C-spaces [2, 9, 10, 14]. Many difficult problems that could not be solved before have been solved by these methods.

If the *start* and *goal* configurations are known *a priori*, and only one (or a very few) queries will be performed in a single environment, then it is generally not worthwhile to perform an expensive preprocessing stage. In this case, a more directed search of the free configuration space could lead to faster solution times (e.g., as opposed to roadmap methods which try to cover the entire freespace). Motion planning methods that operate in this fashion are often called *single shot* methods.

One of the first randomized planning methods is the Randomized Path Planner (RPP) of Barraquand and Latombe [3], which is a single shot planner. This potential field method uses random walks to attempt to escape local minima. In general, such methods can be quite effective when the C-space is relatively uncluttered, but there exist simple situations in which they can fail [10].

Some success has been achieved in adapting the general PRM strategy to solve single queries by restricting attention to ‘useful’ regions of C-space [9, 14]. A related idea is to use a sample of free points to specify promising subgoals [4, 8, 7]. Other approaches that have been used are to limit the part of the C-space that is explored. The planner in [11] finds paths for six-dof manipulators using heuris-

^{*}This research supported in part by NSF CAREER Award CCR-9624315 (with REU Supplement), NSF Grants IIS-9619850 (with REU Supplement), EIA-9805823, and EIA-9810937, and by the Texas Higher Education Coordinating Board under grant ARP-036327-017. Vallejo is on leave from Universidad de las Américas-Puebla, Mexico, and is supported in part by a Fulbright-CONACYT scholarship.

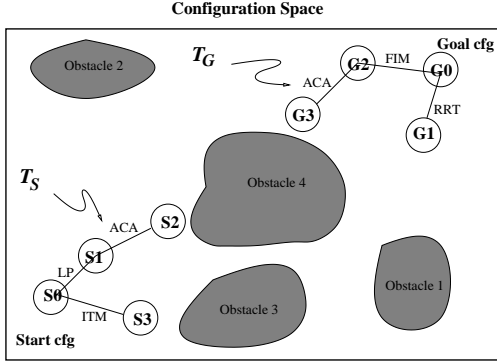


Figure 1: A graph generated during Single Shot planning. Each edge is labeled with the planning algorithm that made the connection.

tic search techniques. A skeleton of the C-space is built incrementally using a local opportunistic strategy in [5]. The Ariadne’s clew algorithm [4] uses genetic algorithms to help search for a path in high-dimensional C-spaces. The Rapidly-exploring Random Tree (RRT) [12] method grows a tree from a specified start configuration.

2 A Single Shot Planning Framework

Although much progress has been made, there are still important classes of problems for which good single shot solutions are needed. In particular, problems in crowded, or cluttered, environments in which the robot must maneuver through tight spots. The difficulty for such problems arises because successful planning requires one to find free configurations in so-called *narrow passages* in the robot’s configuration space.

Our single shot motion planning architecture is an iterative process which evaluates certain characteristics of the current sub-query then selects an appropriate local planning method from the bank of individual single shot motion local planning algorithms (the individual local planners are presented in Section 3). The goal is to allow the best suited local planning methods to be executed for the current situation and to enable them to be used in a cooperative manner. Anywhere from one to all of the local planners in the bank may be executed for a single query.

The general idea of the framework is the following (see Figure 1). To find a path from the Start to the Goal configuration, we grow a tree T_S from the Start configuration and another tree T_G from the Goal configuration. In each iteration, we attempt

```

Repeat
  <qp,alg> := ExtractBest(PQ);
  PartialPath := Alg(qp);
  if ( qp.start in T_S)
    T_S := T_S + PartialPath; T := T_S;
  else T_G := T_G + PartialPath; T := T_G;
  if (T_S != T_G)
    last := LastCfg(PartialPath);
    for each (cfg in T)
      qp1 := <last,cfg>;
      qp2 := <cfg,last>;
      for each (alg in AlgoBank)
        insert( <qp1,alg>, score(qp1,alg), PQ);
        insert( <qp2,alg>, score(qp2,alg), PQ);
  until ( T S == T G )

```

Figure 2: Steps executed during the iteration process of the Single Shot planning.

to generate a path that connects T_S and T_G , and therefore also the Start and Goal configurations.

In each iteration, we consider all potential query pairs with one configuration in T_S and the other in T_G , and all the local planners in the bank, and select the “best” $\langle \text{QueryPair}, \text{Algorithm} \rangle$ tuple from a priority queue, i.e., we select the query pair that is most likely to be connected by a particular local planning method in the bank. (Initially, two query pairs are considered: (Start,Goal) and (Goal,Start)). Then, the selected local planner is executed on the selected query pair and the (partial) path returned is added to T_S or T_G . If the sub-query fails to connect T_S and T_G , then we generate query pairs of the form (last, cfg) and (cfg, last), where last is the last configuration of the partial path returned by the sub-query, and cfg is a configuration in a different connected component from last. Each such pair is evaluated, and appropriate $\langle \text{QueryPair}, \text{Algorithm} \rangle$ tuples are added to the priority queue. The process continues in this way, growing T_S and T_G , until they can be connected or until some maximum number of iterations is reached. A pseudocode explanation of the steps performed in each iteration is shown in Figure 2.

2.1 Evaluation Criteria

The most important, and challenging, operation in each iteration is selecting the “best” $\langle \text{QueryPair}, \text{Algorithm} \rangle$ tuple. This is a multi-step process. First, we calculate characteristics of the potential query pairs using routines in our evaluation bank. Second, based on the query pair and algorithm characteristics, a score is assigned to each $\langle \text{QueryPair}, \text{Algorithm} \rangle$ tuple. Then, we select the “best” $\langle \text{QueryPair}, \text{Algorithm} \rangle$ tuple based on these scores.

For a given query pair, evaluation criteria are needed to characterize: the **local properties** of the space near the start and goal configurations and the **global properties** of the space between them.

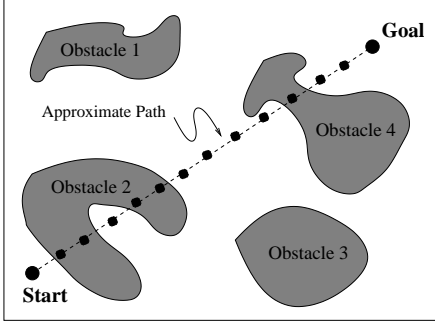


Figure 3: The approximate path P .

Each algorithm in the bank has preset **planner characteristics** (strengths and weaknesses corresponding to those of the configurations) used for matching algorithms to query pairs.

Local Properties: The evaluation begins by determining general, low-level characteristics of individual configurations. For each characteristic property, we compute a numerical value (normalized to $[0, 1]$). Currently, for each configuration c , we consider the following local properties:

- L_1 : Clearance (distance to nearest obstacle),
- L_2 : Translation (free translational space near c),
- L_3 : Rotation (free rotational space near c),
- L_4 : Free (free space near c).

Global Properties: Next, the space between the `QueryPair` configurations c_1 and c_2 is evaluated. This information will help us infer the type of planning required for the query (i.e., general obstacle avoidance, narrow corridor navigation, etc.). This is done in part by evaluating configurations sampled (at a coarse resolution) on a straight-line connecting c_1 and c_2 (an *approximate path* P , see Figure 3). Currently, the global properties (normalized to $[0, 1]$) considered are:

- G_1 : Distance between c_1 and c_2 .
- G_2 : Proportion of free configurations in P ,
- G_3 : Number of obstacles in collision with P .

Planner Characteristics: To facilitate the matching of query pairs with planning algorithms, we assign each algorithm values corresponding to the query pair characteristics which describe the ideal situation in which the planner should be used.

These values are currently assigned using experience and informed guesswork. In the near future the values will be chosen based on simulation results.

2.2 Scoring

The score for a `<QueryPair, Algorithm>` tuple is a weighted average in which we give decreasing importance to the local properties of the start configuration (S), the global properties of the query (Q), and the local properties of the goal configuration (G).

First, for each component (S, Q, G), we sum the differences (in absolute value) between the computed `QueryPair` value and the algorithm’s assigned value.

The score for the `<QueryPair, Algorithm>` tuple $\langle (s, g), A \rangle$ is the weighted average:

$$\text{score}((s, g), A) = S(s, A) + \frac{1}{2}Q(s, g, A) + \frac{1}{4}G(g, A)$$

Note that scores are non-negative with the best possible score being zero (representing a perfect match between `QueryPair` and `Algorithm`).

3 Single Shot Algorithm Bank

The algorithm bank should include a diverse set of algorithms so that at least one of them will perform well in each possible situation. (Recall that we do not require any of the algorithms to be good in all situations, since we will use them in concert to solve the query.) In this section we briefly describe the algorithms available in our prototype implementation.

3.1 Simple ‘Local Planning’ Methods

The first algorithms in our algorithm bank are methods that are commonly used in PRMs as local planners. That is, they are relatively simple, deterministic methods that are fast but not very powerful. The particular methods currently in our bank are: straight-line in C -space, rotate-at-s (for rigid bodies), and simple A^* -like planners. All methods are described in detail in [1].

3.2 Directed Expansion Methods

Each planning algorithm in this section attempts to grow a connected component of configurations from the `start` to the `goal`. The first set of methods (ISM, ITM, IRM) is designed for navigation through narrow C -space corridors where large movements are not possible. The last method, ACA [4], has similar strengths, but does not explore the space as methodically as the I-M methods and so is suitable for slightly less cluttered spaces.

ISM, ITM, and IRM. The general idea of the *iterative spread method* (ISM) is to ‘spread’ away from the **start**, and towards the **goal** when possible. The ITM and IRM methods are similar to ISM, except only translational or rotational movements are allowed, respectively.

Ariadne’s Clew algorithm (ACA). Our bank includes a variant of the Ariadne’s Clew algorithm ([4]), which uses random walks instead of the originally proposed Manhattan paths as its local planning method.

3.3 Random Expansion Methods

The planning algorithms in this section randomly grow a connected component of configurations from the **start** configuration, i.e., they do not consider the **goal**. They will be used mainly in very crowded situations when there is not a good match between any of the other algorithms and the query pair.

Random Walk (RWM). This method performs a random walk of a given number of steps from a given **start** configuration. Its purpose is to enable one to escape from “local minima.”

Rapidly-exploring Random Tree (RRT). Our bank includes a simple version of the RRT algorithm proposed in [12].

3.4 Subgoal Generation Methods

The algorithms in this section generate subgoals for navigating around obstacles. In particular, they are suitable candidates when at least one of the **start** or the **goal** is in un-cluttered space and when the space between them contains obstacles.

First Intersection Method (FIM). The idea behind FIM is to try to go directly from the **start** to the **goal** using the straightline planner. If an intersection occurs, it samples configurations around the obstacle in a manner similar to that used by the OBPRM [2] planner.

Recursive Midpoint Method (RMM). RMM effectively breaks down the initial motion planning query into recursively smaller queries until each sub-query results in a direct collision free connection. The basic idea is that the midpoint mpt of the $(\mathbf{start}, \mathbf{goal})$ segment serves as a subgoal in the query.

4 Experimental Results

In this section we examine the performance of our prototype implementation of our single shot planner. We first consider queries in some relatively simple artificial environments whose successful solution

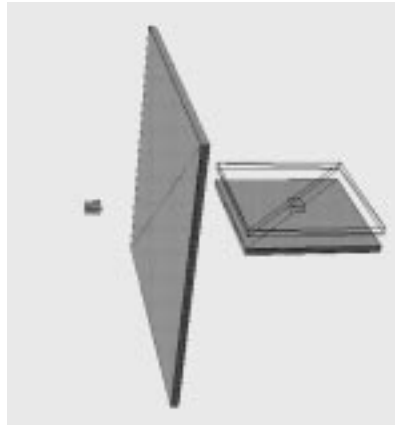


Figure 4: Test Environment Sandwich-Wall

requires the path to pass through different types of regions. We next consider some difficult queries involving complex mechanical CAD models. We compare the single shot planner with an obstacle-based probabilistic roadmap method (OBPRM), and with versions of the rapidly-exploring random tree method (RRT) and the Ariadne’s Clew Algorithm (ACA) which we implemented as subroutines in our algorithm bank (see Section 3).

4.1 Artificial Environments

Sandwich-Wall-Block Our first environment, sandwich-wall, consisted of three blocks, two parallel to each other, and the third perpendicular to the other two. The robot is a smaller block that barely fits in the corridor between the parallel blocks (see Figure 4). The query has the **start** in the corridor between the parallel blocks and the **goal** on the far side of the perpendicular block. As can be seen from the results in Table 1, the single shot method outperforms the others tested. Moreover, it did so by adaptively selecting different planning methods: it first used the LP to go from the **goal** towards the wall, and then used the FIM to go around the wall. The choice of FIM seems a good one as the OBPRM planner, of which FIM is a simple variant, also solved the problem quickly.

Stairs-Block The stairs environment consisted of three identical, parallel blocks, which were slightly offset in a relatively tight bounding box. The robot was a smaller block that barely fits in the corridor beneath the middle block, while there is a bit more clearance available between the top and the middle blocks (see Figure 5). The query has the **start** in the upper right corner, and the **goal** in the upper left in the narrow corridor over the middle block. The single shot method again

Artificial Environments			
<i>Env.</i>	<i>Method</i>	<i>Time (sec)</i>	<i>Solved Query</i>
Sandwich Wall	OBPRM	235	yes
	RRT	439	yes
	ACA	411	yes
	SingleShot	124	yes
Stairs	OBPRM	211	yes
	RRT	2,349	NO
	ACA	2,000	yes
	SingleShot	1,258	yes
Walls	OBPRM	184	yes
	RRT	579	yes
	ACA	262	yes
	SingleShot	210	yes

Table 1: Artificial Environment Results

performs well (Table 1), second only to OBPRM. It solved the query using FIM, LP, FIM, and the ACA methods. Different planners were selected depending on the current **start** and **goal**: FIM was used to move the **start** close to the narrow corridor beneath the middle block, and from there ACA was able to solve the query.

Walls-Stick The walls environment consisted of a series of six parallel walls, four of them having small cut-outs, creating a series of narrow corridors. For this environment, our query only requires the robot (a long, thin stick) to go through one of these narrow openings (see Figure 6). As seen in Table 1, the single shot method is again the second fastest to solve the query (after OBPRM). It used the LP method in both directions, and then the FIM method to travel through the opening. It chose the FIM planner (related to OBPRM) to navigate around/through obstacles.

4.2 Complex CAD models

An important design criteria for complex mechanical systems is that certain parts are easy to service and/or replace. Motion planning algorithms for testing such conditions automatically during the design process would be a great aid to engineers, and have been the subject of previous work [6].

Alpha Puzzle. We considered an *alpha* puzzle environment containing two tubes twisted into an α shape (1008 triangles per tube). One tube is the obstacle and the other the moving object (see Figure 7). The objective is to separate the intertwined tubes. The puzzle can be made easier by scaling the obstacle tube in one dimension, which has the effect of shrinking or widening the gap between the two

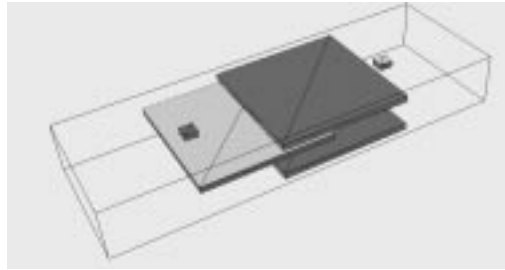


Figure 5: Test Environment Stairs

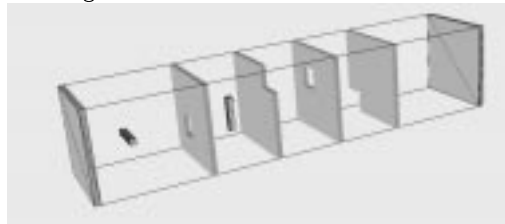


Figure 6: Test Environment Walls-Stick

CAD Environments			
<i>Env.</i>	<i>Method</i>	<i>Time (sec)</i>	<i>Solved Query</i>
Alpha1.5	OBPRM	8,774	yes
	RRT	25,555	yes
	ACA	1,396	yes
	SingleShot	1,396	yes
Alpha1.2	OBPRM	94,150	NO
	RRT	49,455	yes
	ACA	83,254	NO
	SingleShot	68,934	yes
Flange.85	RRT	17,275	NO
	ACA	2,435	yes
	SingleShot	4,837	yes

Table 2: CAD Environment Results

prongs of the α . We applied the algorithms to versions where the obstacle tube was scaled by a factor of 1.5, and 1.2 in the z -dimension. The results are shown in Table 2. The times shown for ACA and the single shot method for the easier version (1.5) are the same since the single shot selected ACA as its first method. However, for the harder case (1.2), ACA could not solve the query, but the single shot did by using some additional methods, namely, ACA twice, and then ITM and finally IRM.

The Flange. The flange environment consists of two objects; the flange and the elbow. The flange is a fairly flat object with a hole in the center (990 triangles). The elbow is a curved object with one extreme wider than the rest of the object (5306 triangles). The start configuration corresponds to the

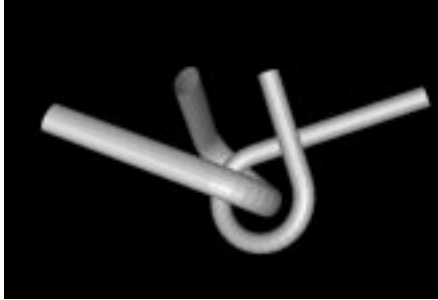


Figure 7: Alpha Puzzle

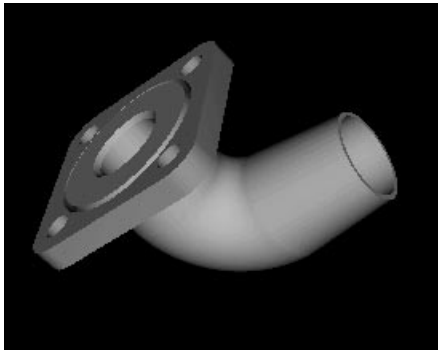


Figure 8: Test Environment Flange

elbow completely inside the flange (see Figure 8), and the goal configuration corresponds to the elbow outside the flange. The problem is to remove the elbow from the flange. Note that this problem is one with a very long and narrow corridor in C-space, requiring a lengthy series of very small translations and rotations of the elbow to successfully remove it from the flange. We consider a version of the original problem with the elbow scaled down to .85 of its original size. The single shot method solved the problem first using ITM, and then the ACA method.

RRT performs well for the alpha environments where there is open space besides the narrow corridor, but it did not perform well in the flange environment where there is little open space.

Acknowledgement

The alpha puzzle was designed by Boris Yamrom of the Computer Graphics & Systems Group at GE's Corporate Research & Development Center. GE also provided us with the Flange environment.

References

- [1] N. M. Amato, O. B. Bayazit, L. K. Dale, C. V. Jones, and D. Vallejo. Choosing good distance metrics and local planners for probabilistic roadmap methods. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 630–637, 1998.
- [2] N. M. Amato, O. B. Bayazit, L. K. Dale, C. V. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, pages 155–168, 1998.
- [3] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *Int. J. Robot. Res.*, 10(6):628–649, 1991.
- [4] P. Bessiere, J. M. Ahuactzin, E.-G. Talbi, and E. Mazer. The ariadne's clew algorithm: Global planning with local methods. In *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, volume 2, pages 1373–1380, 1993.
- [5] J. Canny and M. C. Lin. An opportunistic global path planner. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 39–48, 1990.
- [6] H. Chang and T. Y. Li. Assembly maintainability study with motion planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1012–1019, 1995.
- [7] P. C. Chen and Y. K. Hwang. SANDROS: A dynamic graph search algorithm for motion planning. *IEEE Trans. Robot. Automat.*, 14(3):390–403, 1998.
- [8] B. Glavina. Solving findpath by combination of directed and randomized search. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1718–1723, 1990.
- [9] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 2719–2726, 1997.
- [10] L. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat.*, 12(4):566–580, August 1996.
- [11] K. Kondo. Motion planning with six degrees of freedom by multistrategic bidirectional heuristic free space enumeration. *IEEE Trans. Robot. Automat.*, 7(3):267–277, 1992.
- [12] J. J. Kuffner and S. M. LaValle. RRT-Connect: An Efficient Approach to Single-Query Path Planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 995–10001, 2000.
- [13] J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.
- [14] M. Overmars and P. Svestka. A probabilistic learning approach to motion planning. In *Proc. Workshop on Algorithmic Foundations of Robotics*, pages 19–37, 1994.