Pareto-Optimal Search over Configuration Space Beliefs for Anytime Motion Planning

Shushman Choudhury, Christopher M. Dellin, Siddhartha S. Srinivasa

***Abstract*—We present POMP (Pareto Optimal Motion Planner), an anytime algorithm for geometric path planning on roadmaps. For robots with several degrees of freedom, collision checks are computationally expensive and often dominate planning time. Our goal is to minimize the number of collision checks for obtaining the first feasible path and successively shorter feasible paths. We assume that the roadmaps we search over are embedded in a continuous ambient space, where nearby points tend to share the same collision state. This enables us to formulate a probabilistic model that computes the probability of unevaluated configurations being collision-free. We update the model over time as more checks are performed. This model lets us define a weighting function for roadmap edges that is related to the probability of the edge being in collision. Our approach is to trade off between these two weights, gradually prioritizing edge length over collision likelihood. We also show that this tradeoff is approximately equivalent to minimizing the expected path length, with a penalty of being in collision. Our experiments demonstrate that POMP performs comparably with RRTConnect and LazyPRM for the first feasible path, and BIT* for anytime performance, both in terms of collision checks and total planning time.**

I. Introduction

A significant computational bottleneck of randomized path planning for robots in high-dimensional spaces is the expensive nature of testing for collisions. For roadmap-based methods [15], this bottleneck is manifested in checking edges, which have multiple embedded configurations. Our work attempts to explicitly minimize the number of collision checks while searching for the optimal path in a roadmap.

For many path planning problems, the execution speedup obtained via the shortest path is often negated by the extra planning effort required to find it. Performing a collision check provides exact information but is computationally expensive. Instead, we use a model of the world to estimate the probability of unevaluated configurations to be free or in collision, as we discuss in Sec. III. We ensure that updating and querying the model is inexpensive. Searching for paths based on collision probability does not guarantee optimality, but may speed up the computation of some feasible path. Furthermore, we develop an anytime algorithm to search for successively shorter paths.

A number of previous works have addressed this problem - using the probability of collision as a heuristic to guide the search over paths to obtain a feasible path[21]; lazily and optimistically searching for the shortest path in a roadmap [3]; probabilistically modelling obstacle locations to combine exploration and exploitation in a hybrid approach [16]; using collision probabilities learned from previous instances to modify the roadmap cost function, and filter out unlikely configurations [22]. We mention further related work on the various aspects of our approach in Sec. II.

All authors are with the Carnegie Mellon Robotics Institute
{shushmac,cdellin,ss5}@andrew.cmu.edu

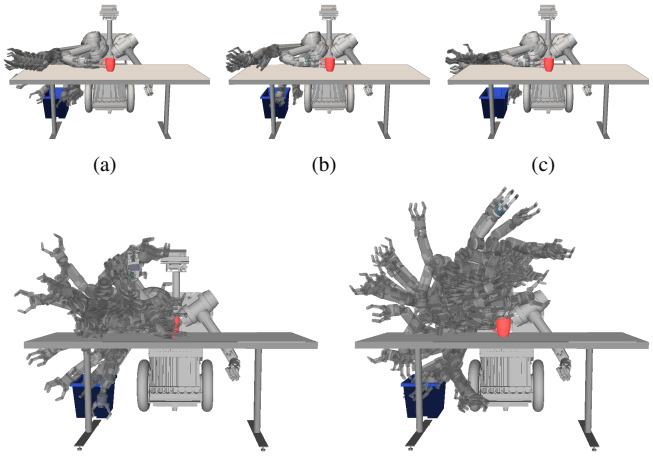


Fig. 1: The first row demonstrates the anytime behaviour of POMP. A sequence of successively shorter trajectories is shown, from the first feasible path obtained in a to the shortest feasible path in c. The second row shows the collision checks required by POMP and RRTConnect respectively to obtain a feasible path. POMP uses a belief model about the configuration space to guide the search for paths that are most likely to be free, and requires fewer collision checks than RRTConnect.

Past work lacks, however, a way to connect the two problems of finding a feasible path quickly and finding the shortest feasible path in the roadmap. We mathematically define the problem we wish to solve, and some related notation, in Sec. IV. Our key insight is that this can be achieved by balancing the probability of collision and the length of a path in the objective, as we show in Sec. V. We name our algorithm POMP, or Pareto Optimal Motion Planner. We show that its behaviour is approximately equivalent to searching for paths of minimum expected cost, with a gradually decreasing penalty of being in collision. The implementation of POMP is outlined in Sec. VI.

In our experiments in Sec. VII, we evaluate the performance of POMP in terms of total planning time and number of collision checks. We compare against RRTConnect [17] and LazyPRM [3] for computing the first feasible path and against BIT* [10] for the anytime behaviour. We run tests over many high dimensional path planning problems. The results demonstrate that POMP performs comparably or better than other widely used algorithms.

Our major contribution is the development of an anytime

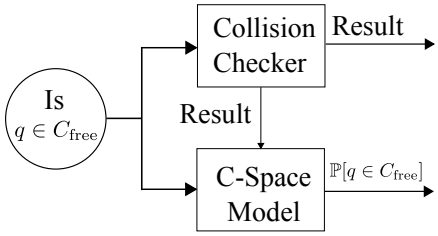


Fig. 2: A schematic diagram of the configuration space belief model.

framework that uses a model of the configuration space to search for successively shorter paths that are likely to be feasible. While there are some limitations, which we mention in Sec. VIII, our algorithm achieves good performance in practice over a number of different planning problems. Figure 1 exemplifies the overall behaviour of POMP.

II. Related Work

For robotic path planning, algorithms have widely used graph-based and sampling-based approaches. Algorithms like Dijkstra's graph search [9] and A* [14] work with some chosen discretization of the problem domain. An alternative approach that has proved effective for high-dimensional systems is sampling-based planning, which randomly samples the problem domain. Two popular methods involve the creation of a graph in state space, as in Probabilistic Roadmaps (PRM) [15] or expanding trees from the start state, as in Rapidly-exploring Random Trees (RRT) [18].

The original probabilistic roadmap approach explicitly evaluated all nodes and edges of the roadmap for feasibility. As a result of the expensive nature of collision-checking, the idea of lazy evaluation of paths became popular. One approach is selecting paths for lazy evaluation based on lowest path length [3], while another is minimizing the probability of the path being in collision, irrespective of length [21]. Our work serves to connect these two ideas under the overall objective of obtaining feasible, successively shorter paths with reduced collision-checking effort.

We reason about both the path length and the probability of collision for an individual candidate path. This analysis is built upon a considerable body of work dealing with bi-criteria path problems. Early work has conducted a systematic study of these problems [13], and devised methods to obtain non-dominated or Pareto optimal paths [7]. It has also provided insights directly relevant to shortest path problems for robots[20].

The probability of collision of a path is derived from an approximate model of the configuration space of the robot. Since we explicitly seek to minimize collision checks, we build up an incremental model using data from previous collision tests, instead of sampling several, potentially irrelevant configurations apriori. This idea has been studied [4, 5] in similar contexts. Furthermore, the evolving probabilistic model can be used to guide future searches towards likely free regions. Previous work has analyzed and utilized this exploration-exploitation paradigm for faster motion planning [1, 16, 22, 23].

Symbol	Description
C	Configuration space
C_{obs}	Obstacle C-Space
C_{free}	Free C-Space
q	Configuration
Φ	C-space model for collisions
$\rho(q)$	Probability of q being collision-free
$G(V, E)$	Graph we search over
w_l	edge weight based on length
w_{m}	edge weight based on probability
$L(\pi)$	path cost based on length
$M(\pi)$	path cost based on probability

TABLE I: Terminology

III. Configuration Space Beliefs

We consider the problem of geometric path planning for a robot with n degrees of freedom. We denote the configuration space (C-space) of the robot by $C \subseteq \mathbb{R}^n$. We denote the set of all configurations in collision by C_{obs}, giving us the free space $C_{\text{free}} \equiv C \backslash C_{\text{obs}}$.

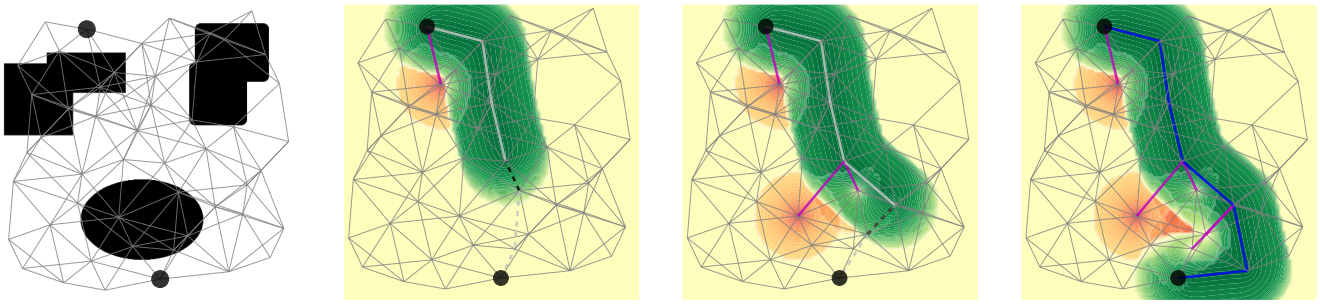
For high dimensional problems, maintaining an explicit representation of C_{obs} (and hence C_{free}) is not computationally feasible. We are interested in regimes where we have an *implicit* representation in the form of a *collision checker*, which takes as input a configuration $q \in C$ and outputs which of the two sets C_{obs} or C_{free} the configuration belongs to. As mentioned in Sec. I, we focus on problem domains where performing each check is computationally expensive.

We observe immediately that the collision checker is an expensive but perfect *binary classifier*, classifying a queried configuration into C_{obs} or C_{free}. We can then formulate an inexpensive but uncertain model Φ that takes in a configuration and outputs its belief that the query is collision-free, represented as $\rho : C \mapsto [0, 1]$. We can build and update this model using a black-box learner [22]. Given a query q, we now have the choice of either inexpensively evaluating $\rho(q)$ from the model Φ, or expensively querying the collision checker. A representation is shown in Figure 2. In this work, we use a k-nearest-neighbour (k-NN) lookup model to estimate the belief of a query. Further details of this are in Sec. VI.

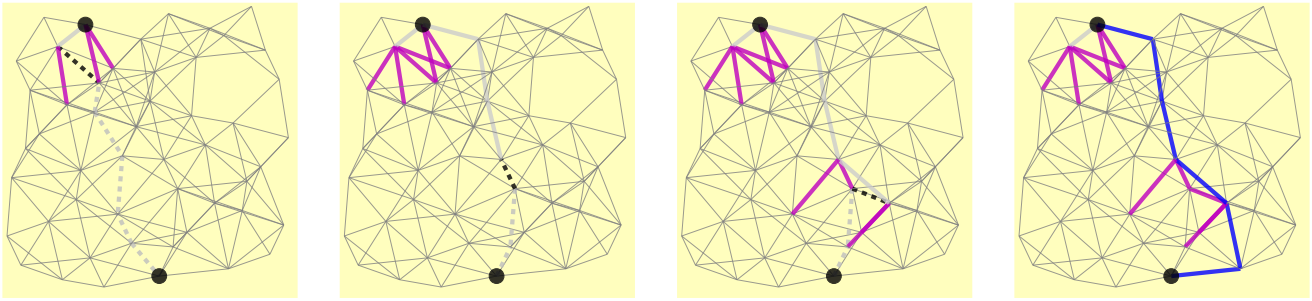
IV. Problem Definition

Following the probabilistic roadmap framework [15], we search for paths on a graph $G = (V, E)$ embedded in C. Given start and target vertices s and t, a *valid* path π in G is a sequence of adjacent edges connecting s and t. A path is *feasible* if every included edge is in C_{free}.

To save computation, although we create the graph explicitly, we *do not* evaluate it apriori, so we do not know if any of its vertices or edges are in C_{free} or C_{obs}. This setup is similar to that of the LazyPRM [3], which then searches over the graph optimistically until it finds the shortest feasible path. We suggest two improvements: (1) to use the belief model to better guide the search to find feasible solutions more quickly and (2) provide an *anytime* algorithm that produces better solutions as the search progresses, eventually finding the shortest feasible path on the roadmap. An illustration for a simple 2D problem is shown in Figure 3.



(a) Problem Environment (b) Model - Iteration 4 (c) Model - Iteration 7 (d) Model - First Path



(e) No Model - Iteration 4 (f) No Model - Iteration 10 (g) No Model - Iteration 15 (h) No Model - First Path

Fig. 3: An illustration of the benefit of using configuration space beliefs. The upper and lower rows show runs of POMP on a 2D planning problem, with some finite model radius and zero radius respectively. The heatmap represents the belief model with green representing the belief of being free, and orange the belief of being in collision. The thin grey edges are unevaluated. The dashed edges are being evaluated. Thick grey edges are evaluated free, and thick magenta edges are evaluated in collision. The blue edges in the rightmost pictures represent the first feasible path in each case. Using the belief model, POMP requires 10 evaluations for the first feasible path, while without the model, it requires 18 evaluations.

A. Edge weights

We define two edge weight functions. The first is $w_l : E \to [0, \infty)$ and measures the length of an edge based on some metric on C. An example metric is the Euclidean distance between the end-points of the edge. For edges that are evaluated to be in collision, the weight is set to ∞. The path length is represented as $L(\pi) = \sum_{e \in \pi} w_l(e)$.

The second is $w_\mathrm{m} : E \to [0, \infty)$, and it relates to the probability of the edge to be collision-free based on our model M. Specifically, $w_\mathrm{m}(e) = -\log(\rho(e))$, where $\rho(e)$ is the probability of e to be collision-free. Note that a known-free edge has $w_m(e) = 0$ and a known-colliding edge has $w_m(e) = \infty$. If we assume conditional independence of configurations given the edge, we can write the log-probability of a path being in collision, $M(\pi)$, in the same summation form as $L(\pi)$:

$$-\log \mathbb{P}(\pi \in C_\mathrm{free}) = -\log \prod_{e \in \pi} \rho(e) = \sum_{e \in \pi} w_\mathrm{m}(e) \equiv M(\pi)$$

We will refer to this M as the *collision measure* of the path. Ensuring that both $M(\pi)$ and $L(\pi)$ are additive over edges enables efficient searches.

B. Weight Constrained Shortest Paths (WCSP)

Our first objective is to obtain some initial feasible path quickly, irrespective of path length. We search for paths that are most likely to be free according to our model. Once we have a feasible path, we search only for paths of shorter length, based on their likelihood of being free. Specifically, we want to search over paths most likely to be free, with a length lower than some upper bound, where the bound reduces over time, with each feasible solution. This can be represented as repeatedly solving

$$\begin{aligned} \hat{\pi} = \ & \underset{\pi}{\text{argmin}} \quad M(\pi) \\ & \text{subject to} \quad L(\pi) < L^* \end{aligned} \tag{1}$$

and subsequently evaluating the returned solution for feasibility. The initial bound $L_0^* = \infty$, after which $L_1^* = L(\hat{\pi}_0)$, where $\hat{\pi}_0$ is the first feasible solution, and $L_2^* = L(\hat{\pi}_1)$ and so on. Therefore, the first iteration of the problem is an unconstrained shortest path problem. For a particular finite upper bound, however, this problem is an instance of the Weight Constrained Shortest Path problem.

V. Approach

We will justify why solving the exact problem we have defined earlier, is not possible efficiently. We then motivate the objective function as a convex combination of both weights. Subsequently, we will show that this is approximately equivalent to searching for a path of minimum expected length.

A. Iteratively solving the WCSP problem

We visualize paths on a 2D plane in terms of their two weights - the path length L and the collision measure M. Each path is a point on this plane, as shown in Figure 4.

For such bicriteria problems, a point (path) is *strictly dominated* by another point if it is worse off in both criteria. For instance, if there are two points γ, γ' such that $L(\gamma') \geqslant L(\gamma)$

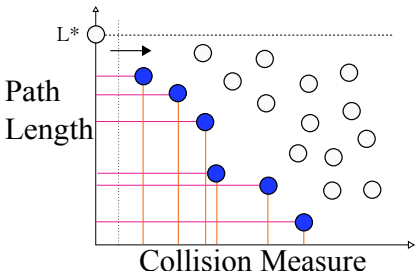


Fig. 4: The LazyWCSP algorithm performs a horizontal sweep on the length(L) - measure(M) plane to select the left-most point. If each point chosen is feasible, and there are no updates to the model, this sweeps out the Pareto frontier of valid points, with respect to initial weights.

and $M(\gamma') \geqslant M(\gamma)$, then γ strictly dominates γ', i.e. $\gamma \succ \gamma'$. A point is *Pareto optimal* if it is not strictly dominated by any other point. The set of Pareto optimal points is known as the *Pareto frontier*.

Consider a simple approach that evaluates paths lazily, called the LazyWCSP method. It repeatedly performs a horizontal sweep over the points in the plane under some horizontal line, the upper length bound (initially there is no line as the bound is ∞). It selects the left-most one (with minimum collision measure) to evaluate, say π_i. If the path is infeasible, it is moved infinitely to the right, and if feasible, it is moved left onto the y-axis, with the collision measure becoming 0. The upper bound L^* is now $L(\pi_i)$, represented by a horizontal line. The collision data of previously unknown configurations updates the model Φ, which in turn updates the x-coordinate of certain points.

Recall that the search for the first feasible path $\hat{\pi_0}$ is an unconstrained shortest path problem with edge weights defined by w_m and lazy evaluation of paths [21]. For the subsequent paths, we have to repeatedly solve the WCSP problem lazily.

There are two major issues with that approach. Firstly, from a practical viewpoint, the WCSP problem is known NP-Hard. There are algorithms for solving it in pseudopolynomial time, by dynamic programming [8] and Lagrangean relaxation [12], but it is highly inefficient to do so repeatedly. Secondly, and more fundamentally, the progress of LazyWCSP does not appropriately address our goal of trading off between path length and collision measure. Consider the scenarios in Figure 5. In 5(b), LazyWCSP would evaluate a point π_1 of marginally lower collision measure and higher path length than another one, π_2. In the general case, where they may be many such points, as in 5(c), this leads to prioritizing several paths that are less promising, i.e. that have a lower gain in length with respect to collision measure.

B. Convex Hull of Pareto frontier

Let us assume that we do not update the model Φ between successive searches; the x-coordinates of unevaluated paths do not change. If an evaluated path is free, it becomes the best feasible solution, otherwise it is removed. Under this assumption, LazyWCSP traces out the Pareto frontier of the feasible paths with respect to their initial coordinates, as shown in Figure 4. In both of the examples in Figure 5, this defers the evaluation of the more promising π_2.

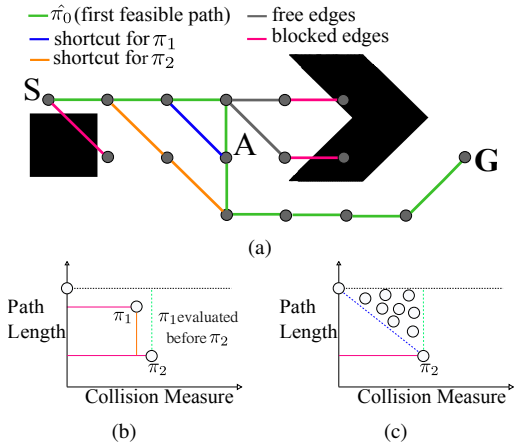


Fig. 5: Two problematic scenarios for using LazyWCSP. A toy case is shown in a, along with two possible corresponding length(L) - measure(M) plots. In b, a small decrement in collision measure for π_1 is prioritized over a larger decrement in path length for π_2. In c, all points above the blue line and to the left of π_2 are evaluated before the more promising π_2. These points correspond to several paths through A.

We can control the tradeoff between the two weights by defining the objective function as a convex combination of the two weights,

$$J^\alpha(\pi) = \alpha L(\pi) + (1-\alpha)M(\pi) \ , \ \alpha \in [0,1]$$

This is the key idea behind our algorithm POMP, or Pareto-Optimal Motion Planner (Algorithm 1). Minimizing J^α for various choices of α, traces out the convex hull of the Pareto frontier of the initial coordinates, as shown in Figure 6(a). The α parameter represents the tradeoff between the weights. Also, optimizing over J^α implicitly satisfies the constraint on L. If the current solution is π_i, then for any path π'

$$\begin{aligned} & J^\alpha(\pi') < J^\alpha(\pi_i) \\ \implies & \alpha L(\pi') + (1-\alpha)M(\pi') < \alpha L(\pi_i) \ [\text{ as } M(\pi_i) = 0] \\ \implies & L(\pi') < L(\pi_i) \end{aligned}$$

The path objective function is additive over edges, so each iteration of the algorithm is now a shortest path search problem. The edge weight for each search is

$$w_j^\alpha(e) = \alpha w_\text{l}(e) + (1-\alpha) w_\text{m}(e) \ , \ \alpha \in [0,1] \qquad (2)$$

When the previous assumption is relaxed, i.e. when collision measures of paths are updated after each search, the corresponding points move and the Pareto frontier moves as well. POMP begins with $\alpha = 0$ and the upper bound $L^* = \infty$, as stated before. After the first feasible solution is obtained, α is increased, and for each value of α, the shortest path search is carried out with the weight function w_j^α. Either POMP finds a feasible path, and the next search uses this path as the current solution, or it does not, and the previous solution is returned. In the latter case, there are no further updates the model can make (as the path returned is fully evaluated), and no other paths can be found with the current α. Therefore the search is restarted after increasing α. A visual description of an intermediate search of POMP is in Figure 6.

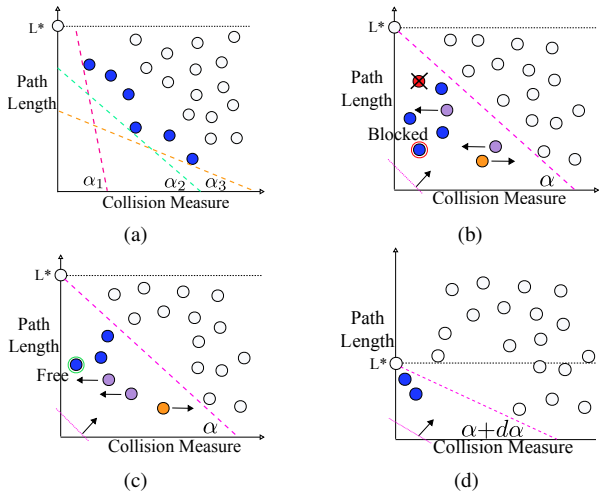


Fig. 6: The key insights to our algorithm. As shown in a, under the assumption of no model updates, the convex hull of the Pareto frontier of the paths is traced out for different values of α. The actual behaviour of POMP in the absence of that assumption is shown through b - d. The diagonal sweep corresponds to searching with J^{α}, and the first point found in the sweep corresponds to the path that minimizes $J^{\alpha}(\pi)$. When a path is evaluated, other paths may have their M-values ■ increased or ■ decreased, and may be ■ deleted if they share infeasible edges with it.

Note that we can use the A* algorithm for the underlying search by scaling the Euclidean heuristic (or any admissible one) by α. Two aspects of the α parameter affect the performance of POMP. One of them is scheduling the values. It can either be increased monotonically from $\alpha' > 0$ to 1 or restarted from $\alpha' > 0$ each time a feasible path is found. We noted empirically that the monotonic increase traced out a sequence of candidates similar to that obtained from restarting, so we chose that to reduce the number of searches. Another issue is the step size of α. The two quantities that are added in the cost function have different units, and the α values need to be chosen so as to balance them reasonably, depending on the problem instance.

C. Minimizing Expected Cost

We will show that the behaviour of POMP is equivalent to minimizing expected path length, with some approximation. This formulation has been used in similar contexts [19].

Let $\hat{J}(\pi) = \mathbb{E}[J(\pi)]$ be the expected length of path π. By the linearity of expectation,

$$\hat{J}(\pi) = \sum_{e \in \pi} \hat{w}_{\mathrm{j}}(e)$$

where $\hat{w}_{\mathrm{j}}(e)$ is the expected length of e. For any edge e, we use a length model where the length of the edge, if free, is $w_l(e)$, and if in collision is $\beta w_l(e)$, where β is a penalty factor ($\beta > 1$). Though we require $\pi \in C_{\text{free}}$, we do not consider a $w_l(e)/\infty$ length model as that would make expected lengths infinite for any unevaluated edges. Because the algorithm eventually evaluates edges, no infeasible paths will be reported as solutions.

$$\hat{w}_{\mathrm{j}}(e) = \rho(e) w_l(e) + (1 - \rho(e)) \beta w_l(e)$$

Algorithm 1 POMP

Input : $G = (V, E)$, w_{l}, w_{m}, s, t, Φ

repeat
  $\pi_0 \leftarrow$ `Dijkstra_Path`(G, w_{m})
  `LazyEvalPath`(G, π_0, Φ)
  if $\pi_0 \in C_{\text{free}}$ **then**
    yield π_0
until $\pi_0 \in C_{\text{free}}$
$\pi_{\text{curr}} = \pi_0$
$\alpha \leftarrow \alpha'$
while $\alpha \leqslant 1$ **do**
  $w_{\mathrm{j}}^{\alpha}(e) = \alpha w_{\mathrm{l}}(e) + (1 - \alpha) w_{\mathrm{m}}(e), \ \forall\ e$
  $\pi_{\text{new}} \leftarrow$ `AStar_Path`$(G, w_{\mathrm{j}}^{\alpha})$
  `LazyEvalPath`(G, π_{new}, Φ)
  if $\pi_{\text{new}} \neq \pi_{\text{curr}}$ and $\pi_{\text{new}} \in C_{\text{free}}$ **then**
    $\pi_{\text{curr}} \leftarrow \pi_{\text{new}}$
    yield π_{curr}
  Increment α
$\pi_{\text{shortest}} \leftarrow \pi_{\text{curr}}$

Algorithm 2 LazyEvalPath

Input : $G = (V, E)$,π, Φ

for $e \in \pi$ **do**
  if e is unevaluated **then**
    `Evaluate`(e)
    Update Φ with collision data for e. This updates w_m for all unevaluated edges.
    if $e \in C_{\text{obs}}$ **then**
      $w_{\mathrm{l}}(e), w_{\mathrm{m}}(e) \leftarrow \infty$ ▷ Known blocked edge.
      return $\pi \in C_{\text{obs}}$
    else
      $w_{\mathrm{m}}(e) \leftarrow 0$ ▷ Known free edge.
return $\pi \in C_{\text{free}}$

The equivalence to Eq. 2 is observed from the following:

$$\begin{aligned}\hat{w}_{\mathrm{j}}(e) &= \rho(e) w_l(e) + (1 - \rho(e)) \beta w_l(e) \\ &= w_l(e)[\rho(e) + (1 - \rho(e))\beta] \\ &= w_l(e)[1 + (1 - \rho(e))(\beta - 1)] \\ &= w_l(e) + w_l(e)(\beta - 1)(1 - \rho(e))\end{aligned}$$

By the Taylor series expansion,

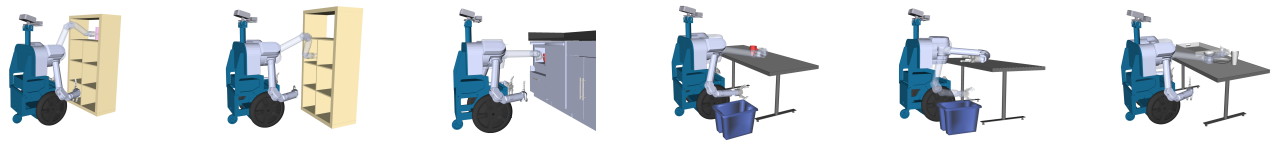
$$log(\rho(e)) = \rho(e) - 1 - \frac{(\rho(e) - 1)^2}{2} + \dots$$
$$\implies -log(\rho(e)) = (1 - \rho(e)) \text{ [neglecting other terms]}$$

Therefore, we obtain,

$$\hat{w}_{\mathrm{j}}(e) = w_{\mathrm{l}}(e) + w_{\mathrm{l}}(e)(\beta - 1) w_{\mathrm{m}}(e) \tag{3}$$

Compare this to $w_j^{\alpha}(e)$ in Eq. 2

$$\begin{aligned}w_j^{\alpha}(e) &= \alpha w_{\mathrm{l}}(e) + (1 - \alpha) w_{\mathrm{m}}(e) \\ &\equiv w_{\mathrm{l}}(e) + \frac{(1 - \alpha)}{\alpha} w_{\mathrm{m}}(e) \text{ [for minimizing]} \\ &\equiv w_{\mathrm{l}}(e) + \gamma w_{\mathrm{m}}(e)\end{aligned}$$



(a) P1 (b) P2 (c) P3 (d) P4 (e) P5 (f) P6

Fig. 7: The test cases we use for our experiments. We name them P1 through P6 for reference. The planning is for the right arm of the robot, which is at the starting configuration in each case. The translucent rendered arm represents the desired goal configuration.

Therefore, the effect of γ is equivalent to that of $w_{\mathrm{l}}(e)(\beta-1)$ as β goes from $\beta' \gg 1$ to 1. γ is constant for all edges, while $w_{\mathrm{l}}(e)$ is different for edges - but $w_{\mathrm{l}}(e)$ is always bounded and finite, so the equivalence between Eq. 2 and Eq. 3 (between $\hat{w}_{\mathrm{j}}(e)$ and $w_j^{\alpha}(e)$) is retained.

Intuitively, β represents the penalty factor that POMP assigns to additional collision checks. Reducing the penalty factor β from $\beta' \gg 1$ to 1 is analogous to increasing α in the earlier formulation from 0 to 1. Both operations represent the increased risk of collision the algorithm is willing to take while searching for edges that, if free, may potentially lead to shorter paths. It should also be noted that at the stage where $\alpha = \beta = 1$, POMP is equivalent to LazyPRM [3].

VI. Implementation

A. Algorithm

POMP is outlined in Algorithm 1. Algorithm 2 is a helper method for lazily evaluating paths. The term **yield** is used instead of **return** to emphasize that the algorithm has anytime behaviour.

B. Configuration Space Model

We have considered the model $\Phi : q \mapsto \rho(q)$ as a black box thus far, because a number of different models exist in the literature [5, 16, 22] which could be used. We utilize a k-NN method similar to one used previously [22].

When $q_i \in C$ is evaluated for feasibility, we obtain $F(q_i) = 0$ if $q_i \in C_{\text{free}}$, 1 otherwise. Then we add $(q_i, F(q_i))$ to the model. Given some new query point q, we obtain the k closest known instances to q, say $\{q_1, q_2 \dots q_k\}$, and then compute a weighted sum of $F(q_i)$ where a weight $w_i = \frac{1}{||q-q_i||}$. Therefore,

$$\mathbb{P}[q \text{ in collision }] = 1 - \rho(q) = \frac{\mathbf{w}.\mathbf{F}}{|\mathbf{w}|}$$

where $\mathbf{w} = [w_1, w_2 \dots w_k]^T$ and $\mathbf{F} = [F(q_1), F(q_2) \dots F(q_k)]^T$.

In principle, the k-NN lookup is $O(\log N)$ while a collision check is $O(1)$. However, for the roadmaps that we tested on, the time for a single collision check was significantly higher than for a model lookup. Asymptotically, the lookup time will exceed check time, which may happen in certain kinds of problems.

VII. Results

We evaluate POMP through a number of experiments on HERB [25], a mobile manipulator designed and built by the Personal Robotics Lab at Carnegie Mellon University. We consider two hypotheses - the benefit of the model for computing the first solution, and the anytime performance.

Our experiments are run on 6 different planning problems for the 7-DOF right arm, shown in Figure 7. The first three problems - P1, P2, P3 - are used for evaluating the first hypothesis. They have goal configurations with significant visibility constraints. The next three problems - P4, P5, P6 - are used for the second hypothesis. Their goal configurations are less constrained than the first set. Thus they have more feasible solutions and better demonstrate anytime behaviour.

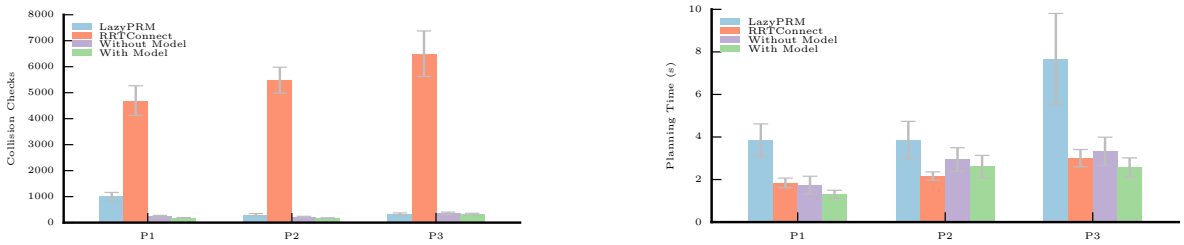
A. Experiments

For each problem, we test POMP over 50 different roadmaps. The distribution of the nodes is generated by Halton sequences [11], which have low dispersion, and the node positions are offset by random amounts. The roadmaps have approximately 14000 nodes, and the r-disk radius for connectivity is 0.3 radians.

Using explicit roadmaps allows us to eliminate all nodes and edges which have configurations in self collision in a pre-processing step, thereby requiring us to only evaluate environmental collisions at runtime. We utilize the same set of default model parameters for each run of POMP - the joint angle resolution is 0.04 radians, the k for k-NN lookup is 15, the prior belief is 0.5, and α increases in steps of 0.1.

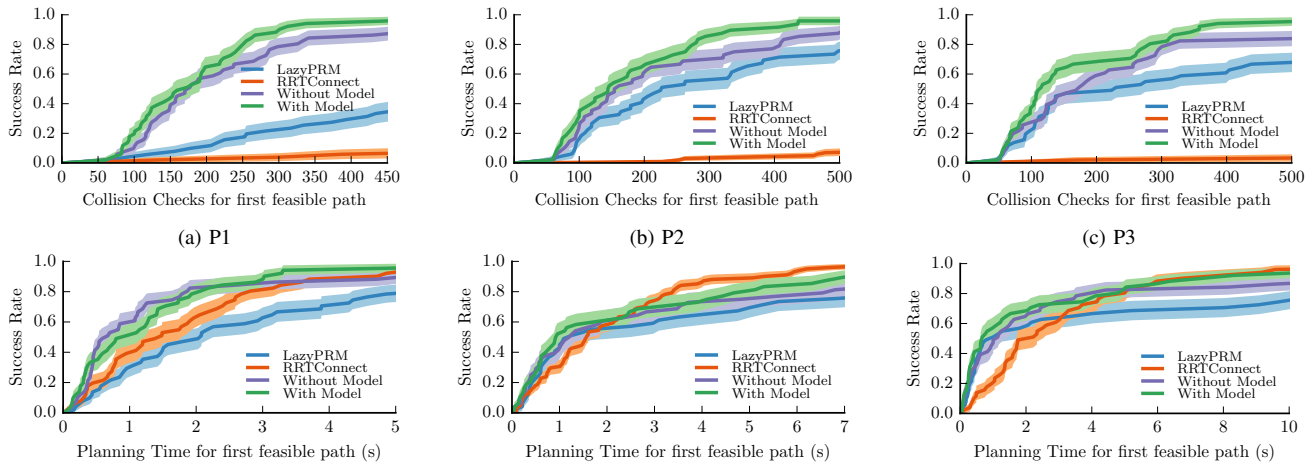
1) Benefit of model for first feasible path: We evaluate the planning time and the number of collision checks required to obtain a feasible solution. We compare against the widely used LazyPRM [3] and RRT-Connect [17]. For RRTConnect, we use the standard OMPL [26] implementation. For LazyPRM, we use the search of POMP with $\alpha = 1$ on the same roadmaps as POMP. We also compare against a variant of POMP that does not use a belief model - it assigns the same probability of collision to all unknown configurations and only sets them to 0 or 1 when they are evaluated. This is to demonstrate the advantage of the model-based heuristic. We name these variants 'With Model' and 'Without Model'.

Figure 8 shows the average collision checks and planning time to compute the first feasible solution for the various algorithms. This is for those roadmaps that have at least one feasible solution for the problem. A second perspective is shown in Figure 9, which shows the success rate of the methods with time and checks. This plot considers all of the



(a) Number of Collision Checks (b) Planning Time (s)

Fig. 8: A comparison of our algorithm POMP with LazyPRM and RRTConnect, in terms of the average planning time and collision checks required for computing the first feasible path. POMP requires far fewer checks than RRTConnect, but spends additional time searching and updating the large roadmap.



(a) P1 (b) P2 (c) P3

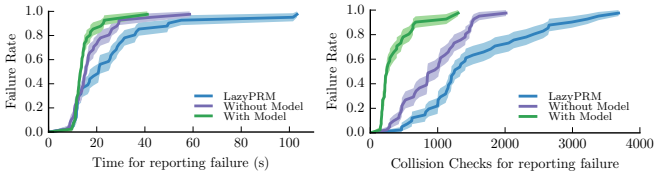
(d) P1 (e) P2 (f) P3

Fig. 9: These plots shows how the collision checks and planning time required varies with the percentage of successful runs for each algorithm. Note that With Model, Without Model and LazyPRM all run on the same roadmap, and would all report a feasible solution if one existed on the roadmap. In each case, the x-axis is cut off after all runs of With Model, on roadmaps with feasible solutions, have concluded. RRTConnect would of course keep searching till it found a solution, and asymptotically its success rate would be 1.

50 roadmaps, whether they have a feasible solution or not, and so the success rate of the methods using them (With Model, Without Model, LazyPRM) all have the same upper bound. The figures show that over all problems, POMP with a belief model shows superior average-case performance. Furthermore, the length of the first feasible path returned by POMP is better than RRTConnect. For the three problems, the average length of feasible paths computed by POMP is approximately 60% that of paths computed by RRTConnect. Additionally, for cases where the roadmap has no feasible solution, POMP using a model reports failure more quickly than the variant without a model and LazyPRM (Figure 10).

An interesting observation from Figure 8 is that though RRTConnect has an order of magnitude more collision checks than POMP, the planning time is still comparable. A qualitative breakdown of the timing shows that POMP spends far less time than RRT-Connect actually doing collision checking. However, it also has far greater overhead for searching the roadmap for candidate paths and updating the collision measure of edges after collision tests.

2) Anytime behaviour: We also evaluate the anytime performance of finding shorter feasible paths over time, up to the optimal path in the roamdap. We compare against BIT* [10] (OMPL implementation), which has demonstrated an



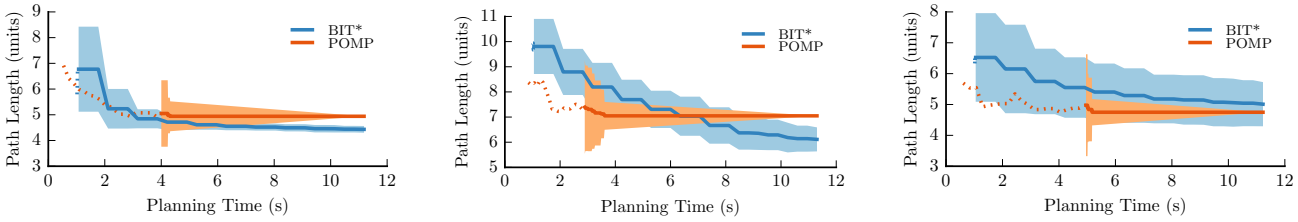
(a) (b)

Fig. 10: POMP with a model reports failure faster than without a model, and LazyPRM, for the same roadmaps and problem instances. This is aggregated over all failure cases from P1 through P3.

anytime performance superior to others. We run tests for 3 different problems P4, P5 and P6, and demonstrate the results in Figure 11. Note that POMP works with only the roadmap provided, without any incremental sampling or rewiring, so the path length does not improve once the shortest feasible path has been obtained. BIT* adds more samples, however, and can continue to obtain improved paths with time.

VIII. Discussion

Given a roadmap constructed apriori, and a black-box configuration space model, POMP efficiently searches for shorter feasible paths in an anytime fashion. We thoroughly evaluated POMP for a set of roadmaps and model parameters and observed consistently good results in comparison to



(a) Anytime - P4 (b) Anytime - P5 (c) Anytime - P6

Fig. 11: A comparison of the anytime performance of POMP with that of BIT*. This is done on 3 separate problems that better demonstrate anytime behaviour than the ones used earlier. The dotted line begins after 50% of the runs have found a solution. The solid line begins after all runs have found a solution. The flattening of the lines for POMP happens after the final roadmap finds the shortest path, as there is no further scope of improvement.

the state of the art. We have shown results for single-query problems, but POMP is also well-suited to multi-query problem instances that enable model re-use.

Like other PRM-based methods, POMP encounters the issue of there not being any feasible path on the roadmap for a particular environment and planning problem. The fast reporting of failure favours beginning with sparse roadmaps and incrementally densifying when no feasible path is found. Given that some approximate model of the world would already be available, techniques like utility-guided sampling [6] could be used to efficiently sample new points in areas where they are likely to be beneficial.

For our implementation of POMP, the C-space belief model uses a simple but effective k-nearest neighbour lookup. We have previously referred to similar models in the literature. Other interesting approaches would be reasoning about the manifolds of the sample points [24] and using persistent homology for occupancy maps [2].

Acknowledgments

This work was (partially) funded by the National Science Foundation IIS (1409003), Toyota Motor Engineering and Manufacturing (TEMA) and the Office of Naval Research. The authors would like to thank the members of the Personal Robotics Lab and Sanjiban Choudhury for valuable discussions and input.

References

[1] O. Arslan and P. Tsiotras. Dynamic programming guided exploration for sampling-based motion planning algorithms. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 4819–4826. IEEE, 2015.

[2] S. Bhattacharya, R. Ghrist, and V. Kumar. Persistent homology for path planning in uncertain environments.

[3] R. Bohlin and L. E. Kavraki. Path planning using lazy prm. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, pages 521–528. IEEE, 2000.

[4] B. Burns and O. Brock. Information theoretic construction of probabilistic roadmaps. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 1, pages 650–655. IEEE, 2003.

[5] B. Burns and O. Brock. Sampling-based motion planning using predictive models. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 3120–3125. IEEE, 2005.

[6] B. Burns and O. Brock. Toward optimal configuration space sampling. In *Robotics: Science and Systems*, pages 105–112. Citeseer, 2005.

[7] J. C. N. Climaco and E. Q. V. Martins. A bicriterion shortest path algorithm. *European Journal of Operational Research*, 11(4):399–404, 1982.

[8] M. Desrochers and F. Soumis. A generalized permanent labeling algorithm for the shortest path problem with time windows. *INFOR Information Systems and Operational Research*, 1988.

[9] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[10] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. *arXiv preprint arXiv:1405.5848*, 2014.

[11] J. H. Halton. Algorithm 247: Radical-inverse quasi-random point sequence. *Communications of the ACM*, 7(12):701–702, 1964.

[12] G. Y. Handler and I. Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 10(4):293–309, 1980.

[13] P. Hansen. Bicriterion path problems. In *Multiple criteria decision making theory and application*, pages 109–127. Springer, 1980.

[14] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

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

[16] R. A. Knepper and M. T. Mason. Real-time informed path sampling for motion planning search. *The International Journal of Robotics Research*, page 0278364912456444, 2012.

[17] J. J. Kuffner and S. M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE, 2000.

[18] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

[19] P. E. Missiuro and N. Roy. Adapting probabilistic roadmaps to handle uncertain maps. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1261–1267. IEEE, 2006.

[20] I. M. Mitchell and S. Sastry. Continuous path planning with multiple constraints. In *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, volume 5, pages 5502–5507. IEEE, 2003.

[21] C. L. Nielsen and L. E. Kavraki. A two level fuzzy prm for manipulation planning. In *Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, volume 3, pages 1716–1721. IEEE, 2000.

[22] J. Pan, S. Chitta, and D. Manocha. Faster sample-based motion planning using instance-based learning. In *Algorithmic Foundations of Robotics X*, pages 381–396. Springer, 2013.

[23] M. Rickert, O. Brock, and A. Knoll. Balancing exploration and exploitation in motion planning. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 2812–2817. IEEE, 2008.

[24] O. Salzman, M. Hemmer, B. Raveh, and D. Halperin. Motion planning via manifold samples. *Algorithmica*, 67(4):547–565, 2013.

[25] S. S. Srinivasa, D. Ferguson, C. J. Helfrich, D. Berenson, A. Collet, R. Diankov, G. Gallagher, G. Hollinger, J. Kuffner, and M. V. Weghe. Herb: a home exploring robotic butler. *Autonomous Robots*, 28(1):5–20, 2010.

[26] I. Sucan, M. Moll, L. E. Kavraki, et al. The open motion planning library. *Robotics & Automation Magazine, IEEE*, 19(4):72–82, 2012.