Motion Planning with Graph-Based Trajectories and Gaussian Process Inference

Eric Huang, Mustafa Mukadam, Zhen Liu, and Byron Boots

***Abstract*— Motion planning as trajectory optimization requires generating trajectories that minimize a desired objective function or performance metric. Finding a globally optimal solution is often intractable in practice: despite the existence of fast motion planning algorithms, most are prone to local minima, which may require re-solving the problem multiple times with different initializations. In this work we provide a novel motion planning algorithm, GPMP-GRAPH, that considers a graph-based initialization that simultaneously explores multiple homotopy classes, helping to contend with the local minima problem. Drawing on previous work to represent continuous-time trajectories as samples from a Gaussian process (GP) and formulating the motion planning problem as inference on a factor graph, we construct a graph of interconnected states such that each path through the graph is a valid trajectory and efficient inference can be performed on the collective factor graph. We perform a variety of benchmarks and show that our approach allows the evaluation of an exponential number of trajectories within a fraction of the computational time required to evaluate them one at a time, yielding a more thorough exploration of the solution space and a higher success rate.**

I. Introduction & Related Work

Motion planning is an indispensable tool that allows robots to navigate in complex environments. Optimal motion planning algorithms attempt to generate trajectories for the robot that are both *feasible* and *optimal* based on performance metrics that may vary depending on the task, robot, or environment. Fast algorithms for motion planing that can be executed in real time are highly desirable, in part so that computational resources can be utilized to perform additional tasks.

Motion planning for robotic systems has primarily been approached using search, sampling, or optimization. Previous work has applied search techniques such as breadth-first search, Dijkstra's algorithm, A*, etc. to robotic motion planning [1]. While informed search techniques like A* or D* can guarantee convergence to the optimal solution [2], these techniques suffer from the exponentially increasing computational costs when applied to problems in higher dimensions (otherwise known as the curse of dimensionality).

Sampling-based algorithms, like PRMs [3] and RRTs [4], [5], allow a good exploration of configuration space and can provide guarantees of probabilistic completeness. However, these approaches often result in non-smooth trajectories that can lead to redundant motion or are inefficient at satisfying tight constraints due to the nature of uniform sampling. Recently, algorithms have been proposed that perform a more informed sampling [6] such that computational resources can be focused appropriately.

Trajectory optimization algorithms start with an initial trajectory and then optimize the trajectory by minimizing an objective function. CHOMP and related methods [7]–[10] optimize a discretized cost functional with covariant gradient descent while STOMP [11] samples noisy trajectories to optimize non-differentiable constraints. TrajOpt [12] optimizes trajectories by formulating a sequential quadratic programming problem. GPMP [13] represents trajectories in continuous time as samples from a Gaussian process (GP) and finds solutions quickly by working with a sparse discretization of the trajectory, while GPMP2 [14] extends this idea to interpret the motion planing problem as probabilistic inference and is able to perform fast inference on sparse factor graphs thereby achieving improved performance and faster computation times compared with previous work.

The drawback of trajectory optimization algorithms is that they can get stuck in poor local optima. Initializing the trajectory as a straight line from start to goal is a common practice, but to improve performance a general strategy is to randomly re-initialize the trajectory and resolve the problem until a successful solution is available or a budgeted computation time runs out [7], [11]–[13]. In practice this strategy may not yield a feasible solution on problems with tight navigation constraints, and the approach can be inefficient when many solution are available but a low cost solution is extremely important to the task.

Cases with multiple feasible trajectories can be understood through *homotopy classes* (see Subsection II-C). In the context of planning, trajectories belong to the same homotopy class [15] when a trajectory can be continuously deformed into another without intersecting any obstacles. Access to information about these classes can allow concentrating resources on finding a solution in a particular class that yields lower cost and also can be useful in exploration with multiple agents where each agent can be assigned a unique homotopy class. However, past work in this area [15]–[17] is limited and requires working out homotopy classes one at a time. This becomes increasingly complicated and inefficient as the dimensionality of the state space increases.

In this work, we propose GPMP-GRAPH, a novel trajectory optimization algorithm that abates the local minima problem by simultaneously optimizing an exponential num-

Eric Huang, Mustafa Mukadam, Zhen Liu, and Byron Boots are affiliated with the Institute for Robotics and Intelligent Machines, Georgia Institute of Technology, Atlanta, GA 30332, USA. {ehuang, mmukadam3, liuzhen1994}@gatech.edu, bboots@cc.gatech.edu. Eric Huang was supported by the Department of Defense (DoD) through the National Defense Science & Engineering Graduate Fellowship (NDSEG) Program. Mustafa Mukadam was supported by NSF CRII Award No. 1464219.

ber of interconnected trajectories. Building on the probabilistic interpretation of motion planning from GPMP2, we first represent a network of interconnected trajectories as a factor graph. In particular, each path in the factor graph corresponds to a valid trajectory. Next, our algorithm leverages the efficiency of inference on structured factor graphs to simultaneously optimize the exponential number of trajectories. Experimental results show that our algorithm gets stuck in fewer local optima and discovers homotopy classes faster than previous optimization-based planners. While our current work is based on the trajectory optimization view of motion planning, it also raises interesting connections to sampling based algorithms (for future work).

II. Background

A. *Gaussian Process Motion Planning*

We begin with a brief review of the GPMP2 motion planning framework, for a full treatment see [14]. In this framework, a continuous-time trajectory is represented as a sample from a vector-valued Gaussian process (GP),[1] $x(t) \sim \mathcal{GP}(\mu(t), K(t,t'))$, with mean $\mu(t)$ and covariance $K(t,t')$, generated by a linear time-varying stochastic differential equation (LTV-SDE)

$$\dot{x}(t) = A(t)x(t) + u(t) + F(t)w(t) \tag{1}$$

where A, F are system matrices, u is a known control input and $w(t) \sim \mathcal{GP}(0, Q_c\delta(t-t'))$ is a white noise process with power-spectral density matrix Q_c [19]. Using the above SDE the mean and covariance of the GP can be specified as

$$\mu(t) = \Phi(t,t_0)\mu_0 + \int_{t_0}^{t} \Phi(t,s)u(s)ds \tag{2}$$

$$K(t,t') = \Phi(t,t_0)K_0\Phi(t',t_0)^\top + \int_{t_0}^{\min(t,t')} \Phi(t,s)F(s)Q_cF(s)^\top\Phi(t',s)^\top ds \tag{3}$$

where Φ is the state transition matrix and μ_0, K_0 are initial mean and covariance. This trajectory is Gauss-Markov and therefore the GP has a sparse precision matrix – a fact later exploited during inference.

Let e be a set of random binary events that are desired, for example, collision avoidance. Then the motion planning problem can be cast as finding the maximum a posteriori (MAP) trajectory

$$x^* = \operatorname*{argmax}_x P(x|e) \tag{4}$$

such that the posterior distribution $P(x|e)$, of the trajectory x given events e, is computed by Bayes rule,

$$P(x|e) \propto P(e|x)P(x). \tag{5}$$

Here $P(e|x)$ is the likelihood of avoiding collision and $P(x)$ is the trajectory prior represented by the GP. Dong et al. [14] showed that this distribution can be efficiently represented by

[1]A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution [18].

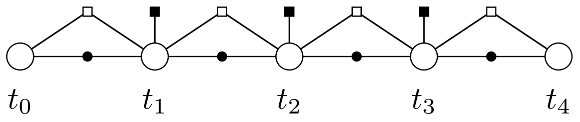


Fig. 1: Example Gauss-Markov chain factor graph in GPMP2 with states (white circle) and factors: GP prior (black circle), collision (black square) and GP interpolation (white square).

a factor graph[2] that allows for efficient inference by solving a nonlinear least square optimization problem while exploiting the graph's sparse structure. The posterior distribution in (5) can therefore be written as

$$P(x|e) \propto \prod_{t_i} f_i^{gp}(x_i, x_{i+1}) f_i^{obs}(x_i) \prod_{\tau=1}^{n_p} f_{i,\tau}^{intp}(x_i, x_{i+1}) \tag{6}$$

with x_i used as a shorthand for $x(t_i)$ and where f_i^{gp} is a binary GP prior factor that connects consecutive states, f_i^{obs} is a unary collision likelihood factor and $f_{i,\tau}^{intp}$ is a collision likelihood factor for a state at τ between any two consecutive support states (at t_i and t_{i+1}) [14]. The state at τ is acquired through GP interpolation i.e Gaussian process regression. A factor graph for an example trajectory optimization problem is illustrated in Fig 1.

Due to the Markovian structure of trajectories, GP interpolation can be performed efficiently because $x(\tau)$ at $\tau \in [t_i, t_{i+1}]$ is a function of only it's neighboring states

$$x(\tau) = \mu(\tau) + \Lambda(\tau)(x_i - \mu_i) + \Psi(\tau)(x_{i+1} - \mu_{i+1}) \tag{7}$$

$$\Lambda(\tau) = \Phi(\tau, t_i) - \Psi(\tau)\Phi(t_{i+1}, t_i) \tag{8}$$

$$\Psi(\tau) = Q_{i,\tau}\Phi(t_{i+1}, \tau)^\top Q_{i,i+1}^{-1} \tag{9}$$

where $Q_{a,b} = \int_{t_a}^{t_b} \Phi(b,s)F(s)Q_cF(s)^\top\Phi(b,s)^\top ds$. The MAP or mode of the posterior distribution under the Laplace approximation of the factor graph provides the optimized trajectory [14].

B. *Sparse Precision Matrix of GP*

The structured GP generated by the LTV-SDE in (1) results in a sparse block tridiagonal precision matrix [20]

$$K^{-1} = A^{-\top}Q^{-1}A^{-1} \tag{10}$$

where

$$A^{-1} = \begin{bmatrix} 1 & 0 & \dots & 0 & 0 \\ -\Phi(t_1,t_0) & 1 & \dots & 0 & 0 \\ 0 & -\Phi(t_2,t_1) & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & 1 & 0 \\ 0 & 0 & \dots & -\Phi(t_N,t_{N-1}) & 1 \end{bmatrix} \tag{11}$$

and

$$Q^{-1} = \operatorname{diag}(K_0^{-1}, Q_{0,1}^{-1}, \dots, Q_{N-1,N}^{-1}) \tag{12}$$

with Φ and Q as defined in the previous subsection, the trajectory going from t_0 to t_N, and K_0 being the initial covariance.

[2]A factor graph is a bipartite graph $G = \{\Theta, F, E\}$, where Θ is the set of variables, F is the set of factors, and E is the set of edges connecting variables to factors.

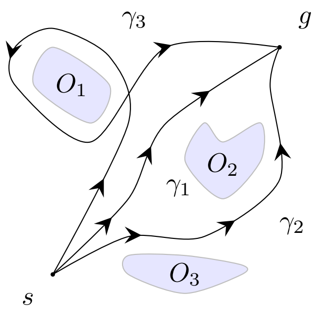


Fig. 2: Example trajectories from different homotopy classes.

C. Homotopy Classes

We use the following definition of homotopic trajectories taken from [15].

Definition 1. *Two trajectories γ_1 and γ_2 connecting the same start and end coordinates are* ***homotopic*** *iff one can be continuously deformed into the other without intersecting any obstacles.*

Given a trajectory γ, the set of all trajectories homotopic to γ form a *homotopy class*. Fig. 2 illustrates an example of distinct homotopy classes. In this example, trajectories γ_1 and γ_2 are non-homotopic because of obstacle O_2. Trajectory γ_3 is non-homotopic to both γ_1 and γ_2 because it loops around obstacle O_1. Note that we can create an arbitrary number of homotopy classes by continuously looping around obstacle O_1. In this work, we will ignore such redundant homotopy classes because our algorithm does not compute them (by design).

III. Graph-based Trajectories from GPs

In this section, we first describe our construction of graph-based trajectories via GPs. Next we show how we can get optimized solution trajectories via MAP inference by generalizing the factor graph structure from Subsection II-A. We introduce the resulting algorithm as GPMP-GRAPH. Then we show how such a construction gives better chances at finding a good local minima on hard problems with tight constraints and in other cases find trajectories in a large number of unique homotopy classes.

A. Net-Graphs

In this subsection, we will incrementally introduce our graphical model for representing a network of trajectories. We start with the Gauss-Markov chain (i.e. the factor graph) used in GPMP2 that employs a GP to represent a distribution of trajectories. In this factor graph, temporally neighboring states are connected via the GP prior factor as shown in Fig. 1. Now consider a graph made of n_c copies of such chains connected to the same start and goal with identical factors in each chain as shown in Fig. 3a. Posterior maximization can be performed on this graph to yield a MAP estimate for each chain. Since each chain is defined by the same GP, the MAP solutions of all the chains i.e. the various local minimas they converge to, is directly influenced by their respective initializations. This procedure would be analogous to resolving the original single chain problem n_c times, where each chain is initialized and solved for, one after another. Since all chains are independent from each other we can write the sparse precision matrix of the GP associated with the full graph as

$$K^{-1} = A^{-\top} Q^{-1} A^{-1} \tag{13}$$

where

$$A^{-1} = \text{diag}(A_1^{-1}, \dots, A_j^{-1}, \dots, A_{n_c}^{-1}), \tag{14}$$

$$Q^{-1} = \text{diag}(Q_1^{-1}, \dots, Q_j^{-1}, \dots, Q_{n_c}^{-1}) \tag{15}$$

and for any chain c_j, A_j^{-1} and Q_j^{-1} are defined as in (11) and (12) respectively.

To simultaneously encode *multiple* trajectories, we add additional edges to this multi-chain factor graph that connect temporally neighboring states across spatially neighboring chains (the spatial neighborhood of a state is based on its initialization). This structure is elaborated through an example in Fig. 3b. We refer to this graphical model as a *net-graph* because it resembles a fishing net. In the net-graph, the new edges connect the states through the same GP prior factor derived from the same LTV-SDE in (1). Every path forward in time now represents a trajectory. Although these trajectories are connected with GP prior factors derived from the LTV-SDE, the marginal paths are no longer sampled from the same GP prior due to the complex interactions between different paths. However, the collective graph is now associated with a new vector-valued GP with a sparse inverse kernel (precision matrix) similar to (13) that still allows for structure exploiting efficient inference. The extra connections made between spatially neighboring chains only fills in the off diagonal blocks in A^{-1} in (14) giving a block tridiagonal structure. Thus the precision matrix remains largely sparse.

B. GPMP-GRAPH

Next, we modify GPMP2 to operate on general graphical models. We call the resulting algorithm GPMP-GRAPH. For ease of explanation, we contextualize the modifications in the framework of the net-graph model below.

To perform inference on the net-graph we define a posterior distribution that factorizes into GP, collision, and GP interpolation factors similar to (6),

$$P(x|e) \propto \prod_{t_i} \prod_{c_j} f_{i,j}^{obs}(x_i^j) \prod_{c_{j'}} f_{i,j,j'}^{gp}(x_i^j, x_{i+1}^{j'}) \prod_{\tau=1}^{n_p} f_{i,j,j',\tau}^{intp}(x_i^j, x_{i+1}^{j'}). \tag{16}$$

Each edge in the graph connects its two states via a GP prior factor, $f_{i,j,j'}^{gp}(x_i^j, x_{i+1}^{j'})$ where x_i^j at t_i and $x_{i+1}^{j'}$ at t_{i+1} are temporal neighbors on chains c_j and $c_{j'}$ respectively that are spatial neighbors (or $j = j'$), each state in the graph has a unary collision factor $f_{i,j}^{obs}(x_i^j)$ and finally, GP interpolation factors $f_{i,j,j',\tau}^{intp}(x_i^j, x_{i+1}^{j'})$ exists between states connected by edges in the graph where GP interpolation can

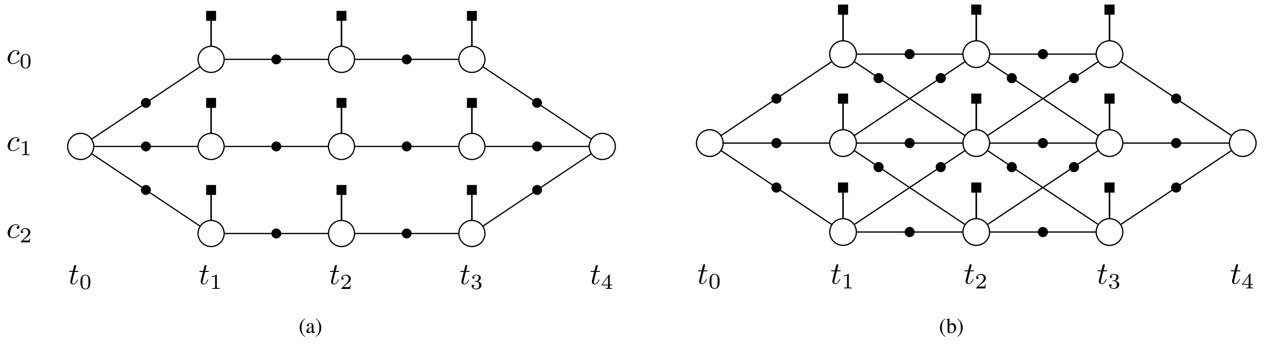


Fig. 3: Example graph-based trajectories constructed from three chains (c_0 to c_2) from start x_0^1 at (t_0, c_1) to goal x_5^1 at (t_5, c_1). Interconnections between chains are absent in (a) and present in (b). States (white circle) are connected temporally in t and spatially in c with GP prior factors (black circle). Collision factors (black square) are shown and GP interpolation factors present between states connected by edges are omitted for clarity.

be performed using (7). This is possible since the Markov property on each edge in the graph is preserved i.e. if two states connected by an edge are observed, any other states between those two states are d-separated from the remaining graph. Finally, inference on the net-graph can be performed through nonlinear least squares optimization to find the mode of the distribution. The inference here too is efficient due to the sparse precision matrix structure of the net-graph as described above.

By performing inference, we evaluate an *exponential number of trajectories* through the edges in this graph much more efficiently than exploring each path sequentially (see Fig. 3 and Fig. 5b). We elaborate on this in Section III-C.

C. Graphs & Exponential Paths

In this section, we show how our net-graph structure has an exponential number of paths by interpreting the net-graph as a directed acyclic graph (DAG). Given a DAG, one can find a topological ordering which divides a DAG into several vertex sets $V_1, V_2, \ldots, V_n$ such that each edge leads from V_i to V_{i+1} for some i. All vertices in V_i must be at time-step t_i for the graph to encode a collection of valid time-series trajectories. We will refer to the number of paths in a graph as its path complexity throughout this paper.

Proposition 1. *A net-graph with n topologically ordered vertex sets has path complexity exponential in n.*

Proof. The net-graph path complexity is bounded by $|V_1| \times |V_2| \times \cdots \times |V_n|$, which is exponential in the number of vertex sets n. This bound is obtained when all adjacent vertex sets form a fully connected bipartite graph. However, note that the path complexity of a general graph is also exponential in n.

Now, a lower bound on the path complexity for net-graphs can be established. To see this, consider the subgraph in Fig. 3 formed by the chains c_0 and c_1 and their interconnected edges. The adjacent vertex sets in this sub-graph are complete bipartite graphs. Therefore, the path complexity of the net-graph is bounded below by 2^{n-2}. In general, the path complexity of a DAG is exponential but the base of the exponent depends on both the number of vertices in each vertex set and the edges between adjacent vertex sets. $\square$

Therefore, the net-graph model allows our planning algorithm, GPMP-GRAPH, to evaluate an exponential number of distinct trajectories simultaneously and efficiently due to its sparse structure (Sec. III-A). The optimized solution from inference can then be evaluated to removes edges that are in collision. The final result is a greater chance of finding a solution (good local minima) on a difficult problem with tight constraints or finding solutions in multiple distinct homotopy classes such that one or many low cost solutions are available for use based on the application. In our experiments, we show how this way of solving problems leads to a higher success rate and better computation times compared to standard straight-line initialization and random restart approaches.

IV. Implementation

We implement our approach using the GPMP2[3] [14] and the GTSAM [21] C++ libraries, which solves the posterior maximization problem as a nonlinear least-squared optimization defined on the factor graph with Levenberg-Marquardt algorithm (initial $\lambda = 0.01$; termination condition: maximum 100 iterations, or relative error smaller than 10^{-4}). Simulations are performed in Matlab for a 2D holonomic robot. In this section we discuss the specific details of our approach.

A. GP Prior

We use a double integrator linear system as the dynamics for our robots with white noise on acceleration,

$$\dot{x}(t) = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} x(t) + \begin{bmatrix} \mathbf{0} \\ \mathbf{I} \end{bmatrix} w(t) \tag{17}$$

where, $\mathbf{0}$ and $\mathbf{I}$ are zero and identity square matrices of size $d \times d$ (d is the degree of freedom of the robot). This follows the construction of the LTV-SDE in (1) and generates a constant velocity GP prior. With this we can write the GP

[3]Available at https://github.com/gtrll/gpmp2

prior factor between any two states connected by an edge in the graph as

$$f^{gp}_{i,j,j'}(x^j_i, x^{j'}_{i+1}) = \exp\left\{-\frac{1}{2}||\Phi(t_{i+1}, t_i)x^j_i - x^{j'}_{i+1} + u_{i,i+1}||^2_{Q_{i,i+1}}\right\} \tag{18}$$

where $u_{i,i+1} = \int_{t_i}^{t_{i+1}} \Phi(t_{i+1}, s)u(s)ds$.

B. Collision Likelihood

We use a hinge loss function h, for the unary collision likelihood factor

$$f^{obs}_{i,j}(x^j_i) = \exp\left\{-\frac{1}{2}||h(x^j_i)||^2_{\Sigma_{obs}}\right\} \tag{19}$$

and the binary GP interpolation factor

$$f^{intp}_{i,j,j',\tau}(x^j_i, x^{j'}_{i+1}) = \exp\left\{-\frac{1}{2}||h(x(\tau))||^2_{\Sigma_{obs}}\right\} \tag{20}$$

where $x(\tau)$ is evaluated from (7). A precomputed signed distance field is used for collision checking and evaluating the hinge loss (see [14] for more details).

C. Homotopy Classes

We use the following algorithm to count the number of homotopy classes given a trajectory set $\mathcal{T}$ and obstacle set $\mathcal{O}$. We assume that the input trajectories are not in collision with any obstacle.

Algorithm 1 Count Homotopy Classes

```
1: function COUNT HOMOTOPY CLASSES($\mathcal{T}$, $\mathcal{O}$)
2:     for $\gamma \in \mathcal{T}$ do $H[\gamma] \leftarrow 0$
3:     $N \leftarrow 0$
4:     for $\gamma \in \mathcal{T}$ do
5:         if $H[\gamma] > 0$ then CONTINUE
6:         $N \leftarrow N + 1$
7:         $H[\gamma] \leftarrow N$
8:         for $\gamma' \in \mathcal{T}$, $\gamma' \neq \gamma$ do
9:             if HOMOTOPIC($\gamma, \gamma', \mathcal{O}$) then
10:                $H[\gamma'] \leftarrow N$
11:    return $N$
12: function HOMOTOPIC($\gamma_1$, $\gamma_2$, $\mathcal{O}$)
13:    $\gamma_2 \leftarrow$ FLIP($\gamma_2$)
14:    for $o \in \mathcal{O}$ do
15:        if POINT IN POLYGON($o, [\gamma_1, \gamma_2]$) then
16:            return False
17:    return True
```

For each trajectory γ with unknown homotopy class, the algorithm finds all the other trajectories which are homotopic to γ and assigns to them an incremental number N. The final count of N is the total number of homotopy classes. We label trajectories γ_1 and γ_2 as homotopic if no obstacle centers o are contained within the polygon formed by joining γ_1 and γ_2. Note, this algorithm only works on 2D trajectories.

Given a net-graph G, we can enumerate and store all the paths as $\mathcal{T}$. The paths in $\mathcal{T}$ are directed (in time) and acyclic because G itself is a DAG.

V. Experimental Results

We compare our method with previous trajectory optimization techniques and sampling-based planners on two simulated datasets.[4] Subsections V-A and V-B respectively introduce the graphical models used to test GPMP-GRAPH and the sampling-based algorithms used for comparison. In the 2D Maze Benchmark (Subsection V-C), we demonstrate an improvement of GPMP-GRAPH over prior techniques for initializing trajectory optimizers. In the 2D Forest Benchmark (Subsection V-D), we show that GPMP-GRAPH outperforms GPMP2 and sampling-based planners on the problem of homotopy class discovery.

A. Graphical Models for GPMP-GRAPH

We use the following graphical models to measure the performance of GPMP-GRAPH. Like previous trajectory optimizers, the baseline graphical model is the straight-line initialization from start to goal. We will refer to this model as *line* in our experiments. The next most common initialization technique is random-restarts, which we will abbreviate as *rr*. In this technique, the optimizer is first initialized with a straight-line and, on failure, re-initialized a maximum n_r number of times with a random trajectory. Our implementation samples the random trajectory from an LTV-SDE GP with power-spectral density matrix Q_r, conditioned on the start and goal states. Note that running GPMP-GRAPH with straight-line and random-restarts is equivalent to using GPMP2.

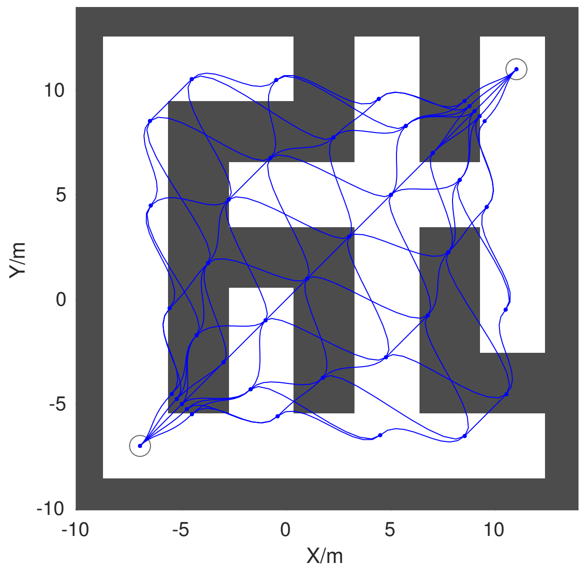
The net-graph model introduced in Subsection III-A has the following parameters. We define n_c to be the number of linear chains connecting the start and goal states. For example, the net-graph in Fig. 3a has $n_c = 3$. Each chain is initialized by perturbing the LTV-SDE mean trajectory (2), in the normal direction by a distance proportional to the GP variance conditioned on start and goal states. An example initialization with $n_c = 5$ is illustrated in Fig. 4a. When computing the GP variance, we use a different power-spectral density matrix Q_n to ensure the perturbed trajectories cover the configuration space. In our results, *ng* will refer specifically to the net-graph having edges connecting all temporally adjacent nodes between spatially adjacent chains (e.g. Fig. 3b).

However, we are also interested in controlling the path complexity of a net-graph. This is achieved by randomly dropping inter-chain edges to form a net-graph. We refer to the graph with fifty random inter-chain edges as *ng-50*. The graph comprised solely of chains is *ng-0*.

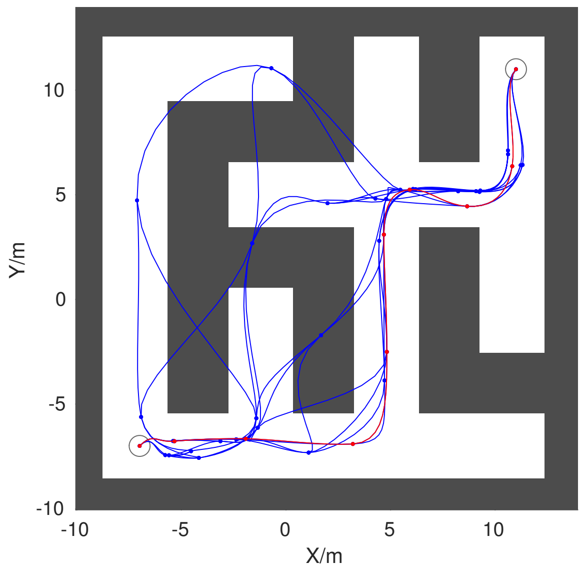
B. Sampling-Based Planners

For comparison, we use a number of popular single-query sampling-based planners in our benchmarks. Specifically, we

[4]A video of experiments is available at https://youtu.be/BTLNtZTMNnA



(a)



(b)

Fig. 4: (a) Example graph initialization and (b) optimized solution to a 4x4 maze (MAP trajectory in red).

TABLE I: Success rates (percentages) for different methods on maze dataset.

maze	ng-50	ng-30	ng-10	ng-0	rr	line
3x3	**97.2**	95.0	89.8	71.9	79.4	51.2
4x4	69.2	**72.5**	63.2	52.8	32.4	18.0
5x5	**35.6**	34.8	30.3	23.3	1.5	0.5

TABLE II: Average execution times (ms) for different methods on maze dataset.

maze	ng-50	ng-30	ng-10	ng-0	rr	line
3x3	59.9	44.1	29.4	22.3	27.1	3.1
4x4	32.2	24.1	16.4	12.4	19.9	4.6
5x5	67.4	52.1	39.1	31.5	8.7	4.9

run RRTConnect [4] (*rrt*) and LBKPIECE [22], [23] (*lbk*), using their OMPL implementation [24].

C. Maze Benchmark

The maze benchmark consists of uniformly sampled perfect mazes, i.e. mazes with a single solution. Each benchmark consisted of 1000 mazes uniformly sampled using Wilson's algorithm [25]. The maze benchmark measures an optimization-based planner's effectiveness at finding the unique collision-free solution in a haystack of local minima.

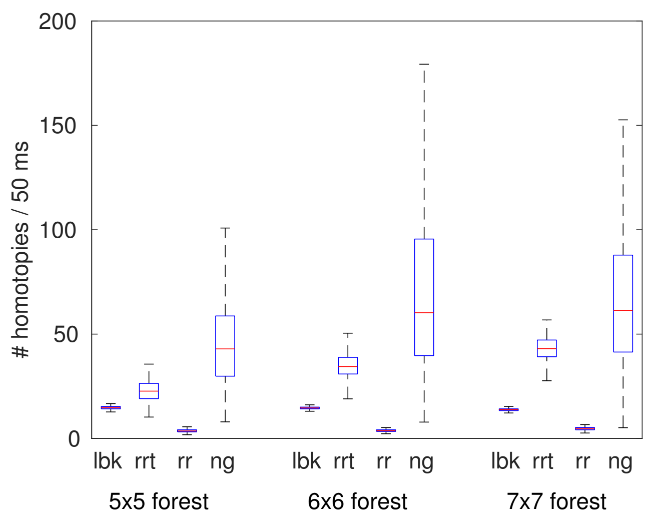
This experiment compares the number of mazes solved for graphical models which contain an increasing number of embedded trajectories. We used straight-line and random-restarts with $n_r = 5$ as the baseline methods. These were compared with a set of net-graphs with $n_c = 5$ chains and 0, 10, 30, 50 inter-connecting edges. Note that each doubling in the number of edges translates to an order of magnitude increase in the number of paths. Each method tested was tuned to the best possible performance on the various maze benchmarks.

The benchmark results are shown in Table I. We omit the sampling-based planner results (100% success rate) from Table I because this experiment aims to measure the improvement of GPMP-GRAPH over prior trajectory optimization methods. The results demonstrate that the percentage of mazes solved increases in proportion to the number of trajectories embedded in the graphical model. Notably, net-graph with 50 edges was able to solve 97.2% of the 3×3 mazes. The benefits of planning using a exponential number of paths becomes more apparent on the larger mazes. On the 5×5 maze, random-restarts only solves 1.5% of the mazes whereas net-graph with 50 edges solves 35.6% of them. Moreover, each order of magnitude increase in path complexity only incurs a 8-15 ms increase in the computational cost of our method (see Table II). An example net-graph before and after optimization using GPMP-GRAPH is shown in Fig. 4.

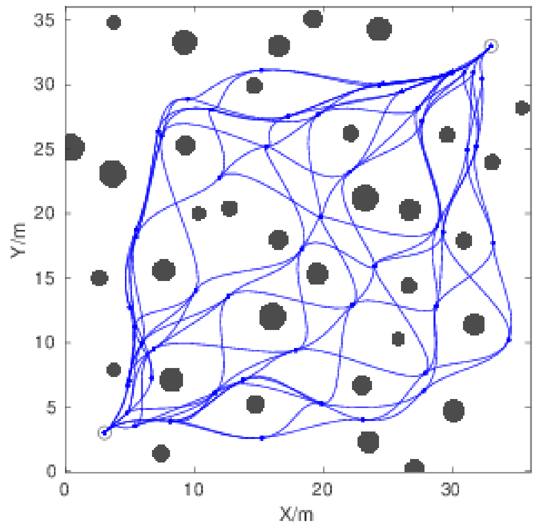
D. Forest Benchmark

The forest benchmark measures the number of homotopy classes found by a planning algorithm. Because homotopy classes partition solution space into distinct sets, it serves as a metric for quantifying the range of solutions generated by a given planning algorithm. For this benchmark, we generate random forests by randomly planting a tree (with random diameter) within each cell of a $n \times n$ grid. The forest benchmarks consist of 300 random forest sets for each forest size of 5×5, 6×6 and 7×7.

We run our trajectory optimization algorithm on this benchmark in the following manner. We initialize GPMP-GRAPH with a net-graph model that has $n_c = 7$ chains and has edges connecting all temporally adjacent nodes



(a)



(b)

Fig. 5: (a) Comparison of sampling-based and optimization-based methods for homotopy discovery on the random forest dataset. (b) An example solution found by GPMP-GRAPH on the 6x6 forest dataset. This particular solution contains 98 homotopy classes.

between spatially adjacent chains (similar to Fig. 3b). Post-optimization, we eliminate all the collision-free trajectories in the optimized net-graph and compute the number of homotopy classes found in 50ms. We use 50ms to normalize the comparisons between GPMP-GRAPH and the single-query planners.

At a high level, we benchmark the single-query planning algorithms by running them multiple times per random forest instance and computing the average number of homotopy classes found in 50ms. In particular, we run random-restarts for a given random forest until the algorithm has been restarted a total of $n_r = 100$ times, while saving each collision-free trajectory. Likewise, we run and aggregate each sampling-based planner 100 times for a given random forest.

We tune each optimization-based planner to the best possible performance for the forest benchmarks. Both random-restarts and net-graph share underlying GP, obstacle factors and path length. The smoothness cost was set to $Q_c = 5$. The obstacle cost was set to $Q_{obs} = 0.3$ with $\epsilon = 1.5$. The path length (i.e. number of nodes in a linear chain) was set to $n = 10$ with a total path time of $t_n = 10$ seconds. Each edge had 4 interpolated obstacle factors. The respective initialization covariances were set to $Q_r = 100$ and $Q_n = 1.35$, respectively.

The forest benchmark results are visualized in Fig. 5a and an optimized net-graph is illustrated in Fig. 5b. From the results, we see that, on average, GPMP-GRAPH finds more homotopy classes than the sampling-based planners or random-restart. However, GPMP-GRAPH also exhibits higher variability compared to the sampling-based planners. We observed two apparent reasons for this variability. First, GPMP-GRAPH tends to fail when the start or goal nodes are blocked by several compactly placed trees with large diameter. In this situation, the control limits (smoothness cost) and the start or goal velocity constrains the trajectories to a cone near the start or goal node. This fights with the collision cost and results in significantly fewer collision-free trajectories because all trajectories must enter or exit through the cone. The naive sampling-based planners do not suffer from this issue because they (crucially) ignore control limits. Second, our net-graph model contains roughly 10,000 trajectories. On select random forests, this enables GPMP-GRAPH to find 2-4 times more homotopy classes than the second best algorithm, RRTConnect, which only generates a maximum of 80 trajectories per 50ms.

VI. Discussion & Future Work

The experiments show that planning with a graph-based trajectory improves performance in terms of success rate and the number of distinct solutions that can be found. In the random forest benchmark, the net-graph structure was able to find a significantly larger set of homotopy classes at a fraction of the previous method's computational costs. In the maze benchmark, we showed that the increased path complexity of the net-graph structure enabled the optimizer to solve more mazes.

While our results show that graph-based trajectory optimization can be a viable technique for avoiding poor local minima, improvements can still be made. The current implementation only considers paths of a fixed length on the graph. A more flexible implementation would reason about paths of variable length, thereby further improving the path complexity of our graph structures.

In future work, it would be interesting to explore connections between trajectory optimization and sample-based motion planning. The factor-graph framework supports incremental updates and thus can efficiently incorporate new samples. Another potentially interesting direction would be to apply our algorithm to finding homotopy classes in higher dimensional spaces.

VII. Conclusion

In this paper, we introduce a novel optimization-based motion planning algorithm, GPMP-GRAPH. This algorithm extends the scope of modern optimization-based planning from optimizing single trajectories to simultaneously optimizing graphs (networks) of trajectories. By leveraging efficient inference on factor graphs, GPMP-GRAPH is able to optimize trajectory graphs which contain an exponential number of embedded trajectories, e.g. the net-graph model. We show through simulated experiments that our algorithm gets stuck in fewer local optima compared to previous trajectory optimizers. Moreover, our algorithm discovers distinct solutions, i.e. homotopy classes, faster than both previous optimization-based and sampling-based planners. Based on our results, we believe that GPMP-GRAPH grows the range of problems that can be solved using optimization-based planning.

References

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

[2] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*. IEEE, 1994, pp. 3310–3317.

[3] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, pp. 566–580, 1996.

[4] 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*, vol. 2. IEEE, 2000, pp. 995–1001.

[5] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[6] 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," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 3067–3074.

[7] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "CHOMP: Covariant Hamiltonian optimization for motion planning," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1164–1193, 2013.

[8] A. Byravan, B. Boots, S. S. Srinivasa, and D. Fox, "Space-time functional gradient optimization for motion planning," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 6499–6506.

[9] K. He, E. Martin, and M. Zucker, "Multigrid CHOMP with local smoothing," in *Proc. of 13th IEEE-RAS Int. Conference on Humanoid Robots (Humanoids)*, 2013.

[10] Z. Marinho, A. Dragan, A. Byravan, B. Boots, G. J. Gordon, and S. Srinivasa, "Functional gradient motion planning in reproducing kernel hilbert spaces," in *Proceedings of Robotics: Science and Systems (RSS-2016)*, 2016.

[11] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 4569–4574.

[12] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.

[13] M. Mukadam, X. Yan, and B. Boots, "Gaussian process motion planning," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 9–15.

[14] J. Dong, M. Mukadam, F. Dellaert, and B. Boots, "Motion planning as probabilistic inference using gaussian processes and factor graphs," in *Proceedings of Robotics: Science and Systems (RSS-2016)*, 2016.

[15] S. Bhattacharya, M. Likhachev, and V. Kumar, "Topological constraints in search-based robot path planning," *Autonomous Robots*, vol. 33, no. 3, pp. 273–290, 2012.

[16] O. Brock and O. Khatib, "Real-time re-planning in high-dimensional configuration spaces using sets of homotopic paths," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 1. IEEE, 2000, pp. 550–555.

[17] E. Schmitzberger, J.-L. Bouchet, M. Dufaut, D. Wolf, and R. Husson, "Capture of homotopy classes with probabilistic road map," in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, vol. 3. IEEE, 2002, pp. 2317–2322.

[18] C. E. Rasmussen, *Gaussian processes for machine learning*. Citeseer, 2006.

[19] S. Anderson, T. D. Barfoot, C. H. Tong, and S. Särkkä, "Batch nonlinear continuous-time trajectory estimation as exactly sparse gaussian process regression," *Autonomous Robots*, vol. 39, no. 3, pp. 221–238, 2015.

[20] T. Barfoot, C. H. Tong, and S. Sarkka, "Batch continuous-time trajectory estimation as exactly sparse Gaussian process regression," *Proceedings of Robotics: Science and Systems, Berkeley, USA*, 2014.

[21] F. Dellaert, "Factor graphs and GTSAM: a hands-on introduction," Georgia Tech Technical Report, GT-RIM-CP&R-2012-002, Tech. Rep., 2012.

[22] R. B. Rusu, I. A. Şucan, B. Gerkey, S. Chitta, M. Beetz, and L. E. Kavraki, "Real-time perception-guided motion planning for a personal robot," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*. IEEE, 2009, pp. 4245–4252.

[23] I. A. Şucan and L. E. Kavraki, "Kinodynamic motion planning by interior-exterior cell exploration," in *Algorithmic Foundation of Robotics VIII*. Springer, 2009, pp. 449–464.

[24] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, http://ompl.kavrakilab.org.

[25] D. B. Wilson, "Generating random spanning trees more quickly than the cover time," in *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '96. New York, NY, USA: ACM, 1996, pp. 296–303. [Online]. Available: http://doi.acm.org/10.1145/237814.237880