Motion Planning with Graph-Based Trajectories as Gaussian Processes

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

***Abstract*— Motion planning as trajectory optimization requires generating trajectories that minimize a desired objective 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 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 nodes 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. 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 task, robot, or environment. Fast algorithms for motion planing that can be used in real time are highly desirable, in part so that computational resources can be utilized to perform other tasks.

Previous work in motion planning has primarily approached the problem from either a sampling-based or trajectory optimization perspective.

Sampling-based algorithms, like PRMs [1] and RRTs [2], [3], 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 to perform more informed sampling [4] 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 [5]–[7] optimize a discretized cost functional with covariant gradient descent while STOMP [8] samples noisy trajectories to optimize non-differentiable constraints. TrajOpt [9] optimizes trajectories by formulating a sequential quadratic programming problem. GPMP [10] 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 [11] extends this idea to interpret the motion planing problem as probabilistic inference and is able to perform fast inference on sparse factor graphs 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 common practice is to randomly re-initialize the trajectory and resolve the problem until a successful solution is available or budgeted computation time runs out. 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*. In the context of planning, trajectories belong to the same homotopy class [12] 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 [12]–[14] 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 present a novel motion planning algorithm that contends with the local minima problem by optimizing many interconnected trajectories simultaneously. Our algorithm can explore multiple homotopy classes efficiently and is less likely than previous trajectory optimizers to get stuck in poor local optima. We leverage the probabilistic interpretation of motion planning in GPMP2, to construct factor graphs that represent a network of trajectories such that each path along the network is a valid trajectory. By performing efficient inference on the factor graph, our algorithm is able to simultaneously optimize an exponential number of potential trajectories in a fraction of time it would take to optimize all such trajectories individually. While our current work is based on the trajectory optimization view

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. Funding source.

Preprint submitted to 2017 IEEE International Conference on Robotics and Automation. Received September 15, 2016.

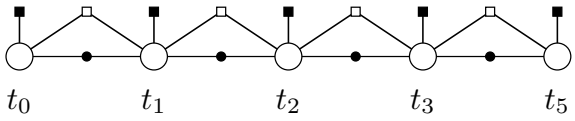


Fig. 1: Example Gauss-Markov chain factor graph in GPMP2 with states (white circles) and factors: GP prior (black circles), collision (black square) and GP interpolation (white square).

of motion planning, it would be interesting to explore the connections to sampling based algorithms in future work and to leverage their potential benefits. In the following sections we detail our approach and validate our hypothesis by performing various benchmarks in simulation.

II. Gaussian Process Motion Planning

We begin with a brief review of the GPMP2 motion planning framework, for a full treatment please see [11]. 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 [16]. Using the above SDE the mean and covariance of the GP can be specified as

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

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

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.

The motion planning problem is cast as finding the maximum a posteriori (MAP) trajectory from the posterior distribution computed by Bayes rule,

$$P(x|e) \propto P(e|x)P(x) \tag{4}$$

where $P(e|x)$ is the collision likelihood and $P(x)$ is the trajectory prior represented by the GP. Dong et al. [11] showed that this distribution can be efficiently represented by 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 (4) can therefore be written as

$$P(x|e) \propto \prod_{t_i} f_i^{gp}(x_i, x_{i+1}) f_i^{coll}(x_i) \prod_{\tau=1}^{n_p} f_{i,\tau}^{intp}(x_i, x_{i+1}) \tag{5}$$

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^{coll} 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}) [11]. 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, GP interpolation can be performed efficiently such that $x(\tau)$ at $\tau \in [t_i, t_{i+1}]$ is a function of only it's neighboring states

$$x(\tau) = \mu(\tau) + \Lambda(\tau)(x_i - \mu_i) + \Psi(\tau)(x_{i+1} - \mu_{i+1}) \tag{6}$$

$$\Lambda(\tau) = \Phi(\tau, t_i) - \Psi(\tau)\Phi(t_{i+1}, t_i) \tag{7}$$

$$\Psi(\tau) = Q_{i,\tau}\Phi(t_{i+1}, \tau)^T Q_{i,i+1}^{-1} \tag{8}$$

where $Q_{a,b} = \int_{t_a}^{t_b} \Phi(b,s)F(s)Q_cF(s)^T\Phi(b,s)^T ds$. The MAP or mode of the posterior distribution under the Laplace approximation of the factor graph provides the optimized planned trajectory [11].

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 Section II. Then we show how these trajectories provide exploration thorough exponential number of initializations simultaneous and efficiently, increasing chances to find good local minima on hard problems with tight constraints. On other problems this approach can find large number of unique homotopy classes as well.

A. Factor Graph on Nets

We start with a Gauss-Markov chain trajectory used in GPMP2 that is a sample from a GP generated by (1). This chain also represents a factor graph where each edge connects temporally neighboring states via the GP prior factor. If we consider n_c such chains connected to the same start and goal with identical factors in the new chains as those specified for the original chain, then each chain is a sample from the same original GP. A posterior maximization performed on this graph results in each of these chains converging to local minima based on their respective initializations. This would be analogous to resolving the problem n_c times each chain initialized one after another.

To simultaneously optimize multiple trajectories, we add edges to the multi-chain factor graph such that these edges connect temporally neighboring states across spatially neighboring chains (the spatial neighborhood of a state is based on it's initialization). This structure is elaborated through

[1] A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution [15].

[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.

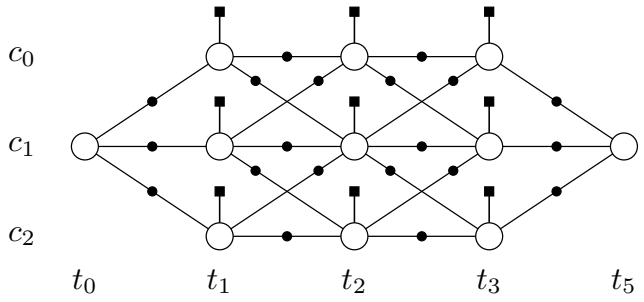


Fig. 2: Example graph-based trajectories constructed from three chains (c_0 to c_2) from start x_0^1 at (t_0, c_1) to goal x_5^1 at (t_5, c_1). States (while circles) are connected temporally in t and spatially in c with GP prior factors (black circle) and collision factors (black square). GP interpolation factors present between states connected by edges are omitted for clarity.

an example in Fig. 2. 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 vector-valued GP with a sparse inverse kernel (precision matrix) that still allows for structure exploiting efficient inference. By performing inference, we evaluate an *exponential number of trajectories* through the edges in this graph much more efficiently than exploring each path in sequentially. (see Figures 1 and 3). We elaborate on this in Section III-B.

To perform inference on the net-graph we define a posterior distribution that factorizes into GP, collision, and GP interpolation factors similar to (5). Each edge in the graph connects its two states via a GP prior factor, $f^{gp}_{i,j,j'}(x_i^j, x_{i+1}^{j'})$ 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 $f^{coll}_{i,j}(x_i^j)$ and finally, GP interpolation factors $f^{intp}_{i,j,j',\tau}(x_i^j, x_{i+1}^{j'})$ exists between states connected by edges in the graph where GP interpolation can be performed using (6). 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 states between the two edges are d-separated from the remaining graph. Finally, inference on the net-graph can be performed through sparse nonlinear least square optimization to find the mode of the distribution efficiently.

B. 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. Clearly, 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. This 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. 2 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 the number of vertices in each vertex set and the edges between adjacent vertex sets.

Therefore, the net-graph model allows the optimization 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 a one or many lowest cost solutions are available for use based on the application. In our experiments we show how this way of solving problems leads to higher success rate and better computation times compared to standard straight-line initialization and random restart approaches.

IV. Implementation

We implement our approach using GPMP2[3] [11] and GTSAM [17] C++ library, 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,

$$\dot{x}(t) = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} x(t) + \begin{bmatrix} \mathbf{0} \\ \mathbf{I} \end{bmatrix} w(t) \tag{9}$$

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

$$f^{gp}_{i,j,j'}(x^j_i, x^{j'}_{i+1}) = \exp\left\{-\frac{1}{2}||\Phi(t_{i+1}, t_i)x^j_i - x^{j'}_{i+1}||^2_{Q_{i,i+1}}\right\} \tag{10}$$

B. Collision Likelihood

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

$$f^{coll}_{i,j}(x^j_i) = \exp\left\{-\frac{1}{2}||h(x^j_i)||^2_{\Sigma_{obs}}\right\} \tag{11}$$

and the binary GP interpolation factor

$$f^{intp}_{i,j,j',\tau}(x^j_i, x^{j'}_{i+1}) = \exp\left\{-\frac{1}{2}||h(x(\tau))||^2_{\Sigma_{obs}}\right\} \tag{12}$$

where $x(\tau)$ is evaluated from (6). A precomputed signed distance field is used for collision checking and evaluating the hinge loss (see [11] for more details).

C. Graphical Models

We provide an overview of the various graphical models used to measure the performance of graph-based trajectory optimization. The baseline graphical model used across all optimization-based motion planners is the linear chain. We consider two variations on the linear chain, namely `straight-line` and `random-restarts`. In the former, the linear chain is initialized using a straight line trajectory in configuration space. In the latter, the linear chain is first initialized using a straight line and re-initialized n_{rr} times with a random trajectory until the optimization returns a collision-free solution. When re-initializing, the random trajectories are sampled from an LTV-SDE GP conditioned on start and goal states. We use a different value for Q_c when sampling the random trajectory to ensure the samples cover the problem's configuration space.

The linear chains are compared with the `net-graph` model which consists of a collection of n_c linear chains sharing start and goal nodes, with edges connecting nodes of time points t_i and t_{i+1} between spatially adjacent chains. Each chain c_i is initialized by perturbing normal to the LTV-SDE mean trajectory (2) by a distance proportional to the conditional GP covariance. Likewise, the LTV-SDE GP used to initialized `net-graph` uses a different value of Q_c to ensure the perturbed trajectories cover the problem's configuration space.

In our experiments, `net-graph` will refer specifically to the graph where there is an edge connecting all temporally adjacent nodes between spatially adjacent chains (e.g. Figure 2). However, it will be of experimental interest to control the path complexity of a graph. This is achieved by randomly removing inter-chain edges from `net-graph`. Thus, we refer to the graph with fifty random inter-chain edges as `net-graph-50`. The graph comprised solely of chains would be `net-graph-0`.

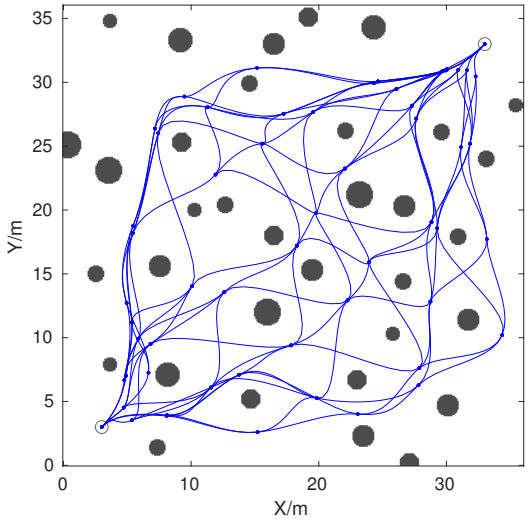


Fig. 3: An example solution found by `net-graph` on the 6x6 forest dataset. This particular solution contains 98 homotopy classes.

D. Homotopy Classes

To compute exact number of homotopy classes on a set of 2D paths, we first enumerated all the collision-free paths in the returned solution. Then for each pair of collision-free paths we compute the winding numbers for all obstacle centers. If all winding numbers are zero, then the two paths are homotopic to each other. The number of homotopy classes follows easily from some additional book-keeping. It is important to note that our method scales with increasing dimensionality of the system and so homotopy classes can be found as easily in higher dimensions. However, currently we have a method to distinguish them only in 2D.

V. Experimental Results

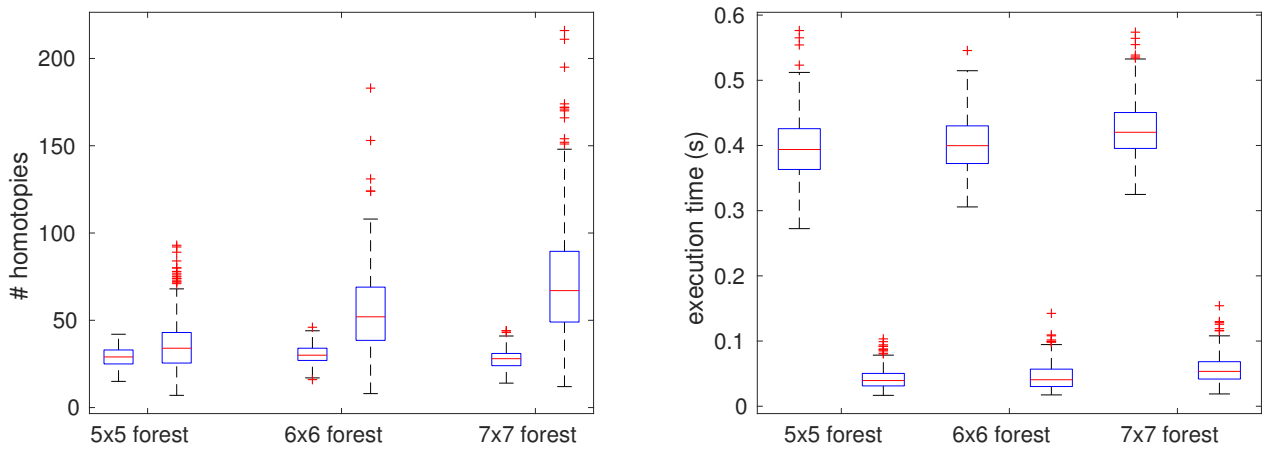
A. Forest Benchmark

The forest benchmarks measure the number of homotopy classes discovered by an algorithm. Homotopy classes partition the space into distinct trajectory sets and thus are a natural metric for measuring the range of solutions an algorithm can generate. This experiment compares the number of homotopy classes found for 1) graphs with linear path complexity using `random-restart` initialization and 2) graphs with exponential path complexity using the `net-graph` initialization.

The forest benchmarks were generated by planting a random tree within each cell of a $n \times n$ grid. The tree location was a random point within each cell and the tree diameter was a random variable between $1/6$ and $2/6$ of the cell width. The set of forest benchmarks consisted of 5×5, 6×6, and 7×7 grid sizes, and each benchmark consisted of 300 randomly sampled forests.

The algorithms were run on each problem in the set of benchmarks in the following way. `random-restarts` was run with $n_{\text{rr}} = 100$ and the homotopy class count was computed on the collision-free paths across all n_{rr} restarts.

Preprint submitted to 2017 IEEE International Conference on Robotics and Automation. Received September 15, 2016.



(a) (b)

Fig. 4: (a) Comparison of `random-restart` (left) and `net-graph` (right) for homotopy discovery on the random forest dataset. (b) Execution times of `random-restart` (left) and `net-graph` (right) during homotopy discovery. Note that `random-restart` was allowed to run an order of magnitude longer than `net-graph`.

`net-graph` was run with $n_c = 7$ and full inter-chain edge connections. Both algorithms shared the same 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 $\epsilon_{dist} = 1.5$. The path length (i.e. number of nodes in a linear chain) was set to 10 with a total path time of 10 seconds. Each edge had 4 interpolated obstacle factors. The respective initialization covariances were set to 100 and 1.35, respectively.

The forest results are visualized in Figure 4. The primary limiting factor for `random-restarts` was finding collision-free paths. We observed a 60% success rate during the restarts. If the obstacles were point obstacles and the robot a point robot, then `random-restarts` would find roughly twice as many homotopy classes; however, such a benchmark would be unrealistic as all sampled paths would be collision free with high probability. There were roughly 10k paths in `net-graph`. Unlike the other method, the success rate was consistently above 99%. The run-time for `net-graph` averaged at 47.7ms and only varied by a few milliseconds across the data-sets. Thus, not only was net-graph nearly an order of magnitude faster than `random-restarts`, it found a significantly higher number of homotopy classes across all benchmarks. Notably, on the 7×7 random forest, `net-graph` would consistently find 2-3 times more homotopy classes than `random-restarts`.

B. Maze Benchmark

The maze benchmark consists of uniformly sampled perfect mazes, i.e. mazes with a single solution. In contrast to the forest benchmark, the maze benchmark measures an algorithms effectiveness at finding the unique collision-free solution in a haystack of local minima. Each benchmark consisted of 1000 mazes sampled using Wilson's algorithm [18].

TABLE I: Success rates 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

This experiment compares the number of mazes solved for graphs which contain an increasing number of paths. We used `straight-line` and `random-restarts` with $n_{\text{rr}} = 5$ as the baseline methods. These were compared with a set of `net-graph`'s with $n_c = 5$ chains and 0, 10, 30, 50 interconnecting 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 were recorded in Table I. The results demonstrate that the percentage of solvable mazes increases in proportion to the number of paths embedded in the graphical model. Notably, `net-graph-50` 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-50` solves 35.6% of them. Moreover, each order of magnitude increase in path complexity only incurs a 8-15 ms increase

Preprint submitted to 2017 IEEE International Conference on Robotics and Automation. Received September 15, 2016.

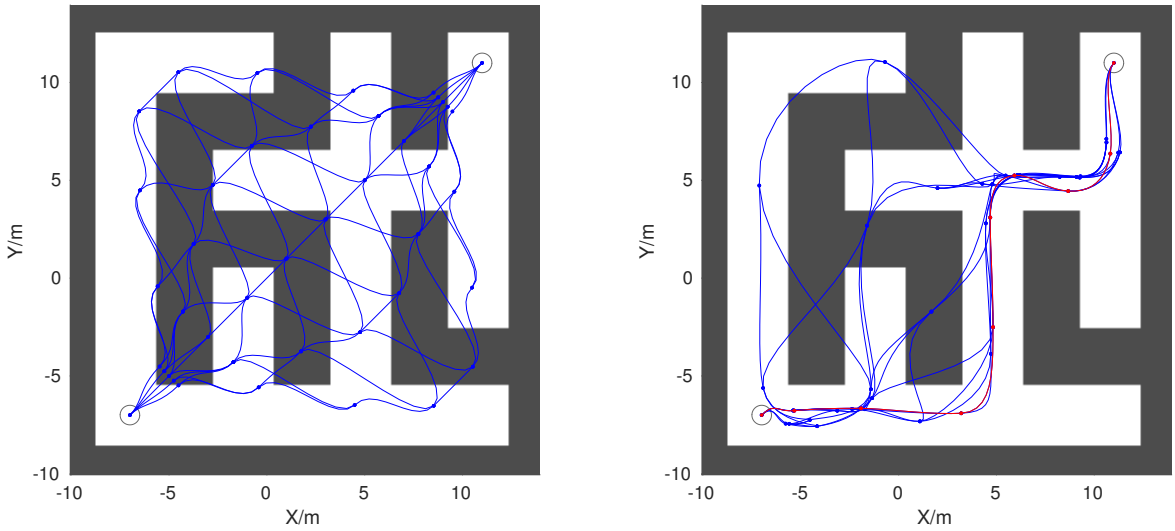


Fig. 5: (a) Example graph initialization and (b) optimized solution to a 4x4 maze (MAP trajectory in red).

in the computational cost of our method.

VI. Discussion & Future Work

The experiments show that planning with an exponential number of paths 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

By using graph-based trajectories we leverage the expressiveness of a factor graph and show it benefits in motion planning as trajectory optimization. This structure allows for exploring exponential number of trajectories with little added computational cost possible due efficient optimization on sparse factor graphs. With benchmarks in simulation we show that our method increases the chances of finding good local minima on hard problems thereby improving success rates. We also show that our method can find a much larger number of homotopy classes in an order of magnitude lesser time than standard trajectory optimization algorithms.

References

[1] 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.

[2] 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.

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

Preprint submitted to 2017 IEEE International Conference on Robotics and Automation. Received September 15, 2016.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

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

[18] 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

Preprint submitted to 2017 IEEE International Conference on Robotics and Automation. Received September 15, 2016.