Planning Planning: The Path Planner as a Finite State Machine

Valentin N. Hartmann[1,2], **Ozgur S. Oguz**[1,3], **Marc Toussaint**[2,3]
[1]Machine Learning & Robotics Lab, University of Stuttgart, Germany
{firstname}.{lastname}@ipvs.uni-stuttgart.de
[2]Learning and Intelligent Systems Lab, TU Berlin, Germany
[3]Max Planck Institute for Intelligent Systems, Germany

Abstract

Path planning is a crucial step in many robotic applications. To solve the planning problem the main approaches are sampling-based and optimization methods, each of which excel in different areas. However, methods combining both areas in a principled way do not exist yet. In this work, we formulate a representation which enable us to formalize various path planning algorithms as state machines, and allows us to combine the strengths of different approaches in a generic framework. Using this formalism, a path planning problem is solved by successively altering a state using various operations (e.g. *sample*, *connect*, etc.) until a solution is found. We demonstrate the framework by constructing policies for the state machine which correspond to widely known algorithms. We present promising results of the framework in the form of manually constructed state machines which combine optimization and sampling based methods, and discuss their advantages and drawbacks. We show that our framework is able to handle complex path planning problems, and is competitive with traditional, more developed approaches. The simplicity of the resulting state machines hints that such policies could be learned, going in a similar direction as architecture search in machine learning.

1 Introduction

Path planning can be seen as a solved problem for low dimensional spaces and simple systems. Various different approaches exist, and computational power to apply them is abundantly available. Problems arise in high dimensional, or dynamically constrained domains. Thus, different decompositions are often proposed, attempting to simplify problems, and solve these first, and use them as guidance for the more complex problem (e.g. (Wagner and Choset 2011) or (Orthey, Escande, and Yoshida 2018)).

A complete discussion of path planning methods is out of scope – we give a brief overview over the most relevant material here. Path planning methods can generally be divided into two large areas: (i) optimization based, and (ii) sampling-based methods.

Optimization based methods have the advantage of being able to incorporate a wide variety of cost functions, for which optimality can be guaranteed under some (usually strong) assumptions. Attempts to compute paths more robustly under a wider range of circumstances have been made e.g. in (Ratliff et al. 2009; Kalakrishnan et al. 2011; Schulman et al. 2013; Herbert et al. 2017). However, none of these approaches are *complete*, i.e., a feasible path might not be found, even if one exists. The reason for this failure mode is usually the non-convexity of the formulated optimization problem, caused by obstacles or other (non-holonomic) constraints.

Sampling-based algorithms such as RRTs (LaValle 1998) and PRMs (Kavraki et al. 1996) are probabilistically complete, and can be made probabilistically optimal (Karaman and Frazzoli 2011). The resulting paths are in general non-smooth, and have to be post-processed for smoothness, dynamic feasibility, and optimality before execution. For this reason, sampling-based methods are often used as initialization or as a backup to a (possibly) faster, but non-complete, optimization based planner. Extensions of the algorithms help with convergence speed and sample efficiency, but do not convincingly solve the problems. In addition, sampling-based algorithms can break down in high-dimensional systems (for a more in-depth discussion, see (Branicky et al. 2001)).

Hybrid search methods combine different paradigms into one algorithm, and could be seen as a third category of approaches: (Choudhury et al. 2016) uses a local optimizer during the `extend` phase of the RRT algorithm, (Luna et al. 2013) present a meta algorithm to merge different (partial) solutions to a single path with shorter length.

Combining both sampling-, and optimization-based methods would be advantageous, but has not been done in the most commonly used frameworks (OMPL (Şucan, Moll, and Kavraki 2012), TrajOpt (Schulman et al. 2013)).

Our key contributions in this paper are

- A unifying representation for various path planning algorithms in the form of a state-machine and a corresponding policy, and
- Examples of policies that induce new algorithms and combine sampling- and optimization based methods.

2 Problem Formulation

In this work, we focus on planning in the configuration space, i.e. we decouple the planning problem into finding an admissible *path*, and in a second step, search for an admissible time-parametrization that satisfies possible dynamic constraints f (for more details on this decoupling, see (Pham, Caron, and Nakamura 2013)).

Hence, we formulate the problem of finding a path $x : [0, T] \to \mathcal{X}_{\mathcal{M}}$ for a robot-model $\mathcal{M}$ as

$$x^*(t) = \underset{x}{\operatorname{argmin}} \int_0^T c(x(t))\, \mathrm{d}t$$
$$\text{s.t. } x(0) = \mathcal{X}_{\mathcal{M},\text{start}},\ x(T) \in \mathcal{X}_{\mathcal{M},\text{goal}}$$
$$\forall t \in [0, T] : x(t) \in \mathcal{X}_{\mathcal{M},\text{free}}, \tag{1}$$

where $\mathcal{X}_{\mathcal{M}} \subset \mathbb{R}^n$ is the full configuration space of the robot $\mathcal{M}$, and $\mathcal{X}_{\mathcal{M},\text{free}} = \mathcal{X}_{\mathcal{M}} \setminus \mathcal{X}_{\mathcal{M},\text{obs}}$ is the free configuration space. In practice, $\mathcal{X}_{\mathcal{M},\text{free}}$ and $\mathcal{X}_{\mathcal{M},\text{goal}}$ are defined by sets of constraints[1], i.e. $g(x(t), t) \leq 0$, and $h(x(t), t) = 0$.

Common examples for the cost function c are path length $c = |\dot{x}(t)|$, or squared acceleration $c = \ddot{x}(t)^2$. In case we only want to *find* a path, we choose $c = 0$, and the problem reduces to a constraint-satisfaction problem.

3 The Planning State Machine

Our approach to solve the problem Eq. (1) efficiently is to sequence and combine various computational operations, and *explicitly combining* sampling- and optimization-based methods. Solving the planning problem thereby becomes a decision process of how to combine the computational operations. We describe the process as a state machine – alternative formulations in terms of (PO)MDPs (as done in (Choudhury et al. 2018) for learning ideal exploration strategies of given graphs) would equally be possible.

We define the *state* $x = (\alpha, D, \mathcal{P})$ as a three tuple of

- the finite state $\alpha \in \mathcal{A}$, which primarily determines which operations are performed and how the process transitions,
- the sample and path data D, which contains all data collected from previous runs of sample-based searches, constraint solvers, or path optimizers (e.g. sampled vertices, connected edges, values of objective functions),
- the problem specification $\mathcal{P}$, which defines the robot model $\mathcal{M}$ and the objectives c, $(h, g)_{\text{start}}$, $(h, g)_{\text{free}}$, $(h, g)_{\text{goal}}$.

In the following sections we give more details on each of these state components.

3.1 $\mathcal{P}$ and Admissible Abstractions

In order to solve the original problem $\mathcal{P}_0$ (with model $\mathcal{M}_0$), it is often useful to first solve simpler problems. Hence, we include the problem description Eq. (1) as part of our state which can be altered. More specifically, to guide solvers of $\mathcal{P}_0$ we may first want to solve easier versions of the problem (e.g. admissible[2] abstractions of $\mathcal{P}_0$). Some examples of such easier problems are:

- Removal of a subset of collision shapes of the robot (e.g., only planning for the base of a mobile robot),
- Removal of kinematic constraints (e.g., making an end-effector free flying),
- Planning for agents separately in a multi-agent setting.

These *abstraction operations* in our state machine modify the state $\mathcal{P}$ of the planning process. Such an abstraction also has to be un-done during the planning process: We will define operations to project abstract solutions back into the original configuration space, thereby guiding the search for solving the original problem.

3.2 D: Storing Preliminary Results

The output of sample-based methods like RRT and PRMs are collections of (feasible) configurations, optionally together with neighborhood information. We store all results generated by such methods in a configuration graph $D_{\mathcal{P}}$, where vertices are feasible configurations $x \in \mathcal{X}_{\mathcal{M}}$ (in the configuration space of $\mathcal{P}$), and edges $e \in \mathcal{E}$ indicate ϵ-near neighborhood. In addition, we label vertices as goal-feasible if $x \in \mathcal{X}_{\mathcal{M},\text{goal}}$. Thereby, $D_{\mathcal{P}}$ also stores goal samples, which are generated by dedicated constraint solvers, and used, e.g., by bi-directional RRTs. In addition, we store generated admissible paths and their corresponding cost in D, i.e. paths that satisfy the constraints in Eq. (1). These initial solutions can be used for e.g. initialization of a possible solution when projecting a path into a higher dimensional space, or for constraining the sampling or the optimization process.

To summarize, $D = \{D\}^i_{\mathcal{P}_i}$, where $D_{\mathcal{P}}$ is the tuple $(\mathcal{G}, \mathcal{Q})$, where $\mathcal{G}$ is a graph of feasible configurations, and $\mathcal{Q}$ is the set of feasible paths.

3.3 α, the Operations, and the Policy π

The finite state α uniquely identifies the operation that is performed in the current state. Every operation may modify D and $\mathcal{P}$, and in addition may give a return value y, e.g., whether a feasible path was found. Based on α, the updated $D, \mathcal{P}$, and the return value, the policy determines the new α':

$$\pi : (\alpha, D, \mathcal{P}, y) \mapsto \alpha', \tag{2}$$

which defines a state transition in our state machine.

In Table 1, we define some possible operations, their parameters, and how they change the state. Concrete planning policies π are described in the next section.

3.3.1 Termination Criterion In general, one would like to know if a given $\mathcal{P}_0$ can be satisfied, and terminate if its infeasibility is guaranteed. In practice, due to the nonlinear constraints, this is not possible. We thus terminate the planning process either if $\mathcal{P}_0$ is satisfied (as is often done e.g. with RRTs), or a computational budget is used up (as is often the case for RRT*). Relaxing the constraint of the computational budget can make a planning policy π probabilistically

[1] $\mathcal{X}_{\mathcal{M},\text{start}}$ is commonly a single configuration x_0, but can be a set of configurations defined by constraints in the same way as $\mathcal{X}_{\mathcal{M},\text{free}}$ and $\mathcal{X}_{\mathcal{M},\text{goal}}$.

[2] A tuple $(\mathcal{P}, \phi)$ is an admissible abstraction of an original problem $\mathcal{P}_0$, if $\phi : \mathcal{X}_{\mathcal{M}_0} \to \mathcal{X}_{\mathcal{M}}$ project from the configuration space of $\mathcal{P}_0$ into the configuration space of $\mathcal{M}$, and $\mathcal{P}$ is a *lower bound* of $\mathcal{P}_0$ (Orthey, Escande, and Yoshida 2018; Toussaint and Lopes 2017). *Lower bound* means that if the abstraction $\mathcal{P}$ is infeasible, then the original problem $\mathcal{P}_0$ must also be infeasible.

Operation	Parameter	Description
Sampling	Sampling strategy l	Sampling the state $\mathcal{X}_{\mathcal{M},\text{free}}$ leads to a new sample in D. Strategies include rejection-, goal-, and informed sampling.
Extend	Goal vertex v_g	`extend` adds a new feasible state, and an edge to D. The vertex which we extend from is the one that is closest to v_g.
Connect	Vertex v, Connection radius r	The `connect`-operation tries to connect to feasible configurations from the state D, and adds edges to the graph.
Prune	pruning strategy q	`prune` labels some edges in the graph (chosen according to q) as invalid.
Optimize	start and goal vertices v_s, v_g, cost function c	`optimize` connects two nodes in D using an optimization approach. It can be seen as specialization of `connect`.
Smooth	cost function c, initial path y	`smooth` is similar to `optimize`, but takes an initial path as additional argument.
Abstract	Abstraction method a	This operation changes the problem description by e.g. abstracting the robot model.
Project	projection method p, vertex v	Projects a given vertex v to a higher of lower dimensional state.
Graph Search	Vertices v_i, v_j	Checks if two vertices in D are connected.

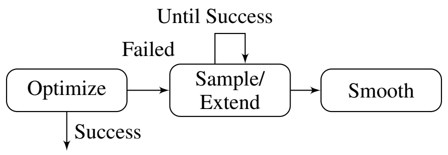
Table 1: Operations, the relevant parameters, and a brief description

complete, however in practice, we want to have the guarantee that the state machine halts.

4 The State Machines

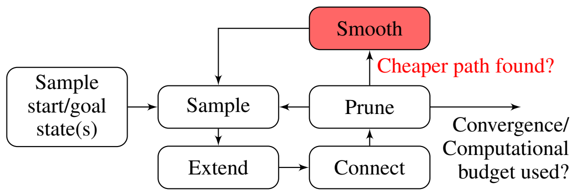
We show examples of commonly known algorithms such as RRT and PRM in Appendix A - this might help clarify some of the concepts introduced before. In the following Section, we describe new combinations of the previously introduced operations to induce new planning algorithms in our framework. In this work, we describe a policy π as a graph, where the nodes describe the actions α_i, and the transitions are determined by π. In the sections where we adapt known algorithms, the changed parts are highlighted in red.

Combined robust solver We first attempt to solve the complete problem using an optimization approach, and switch to a complete solver (i.e. either PRM or RRT in this case) if the optimizer was unsuccessful:



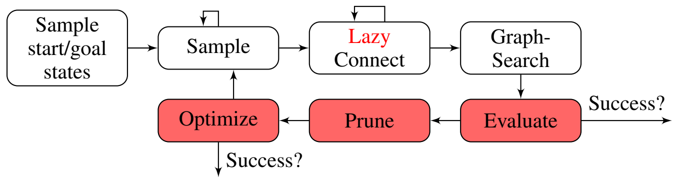
The path from the RRT solver then has to be smoothed. This approach has the advantage of a more robust path planner, while keeping the advantage that a solution will be found quickly by the optimization approach in some cases.

Informed RRT*-Smooth Informed RRT* changes how samples are generated based on knowledge on the paths found so far. However, it is inefficient in improving the bound it uses because it limits itself strictly to sampling based methods. We introduce an additional step here, namely using a smoother on the path that was found to improve the cost bound that is used for sampling.


Smooth
Cheaper path found?
Sample start/goal state(s)
Sample
Prune
Convergence/ Computational budget used?
Extend
Connect

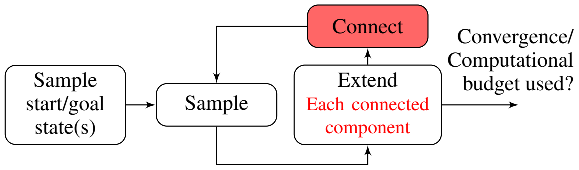
This means that we are using RRT explicitly as path planner to discover paths in different homotopies, and assume that improving the path using sampling-based methods is wasteful compared to optimization based methods. Here, we invoke the `smooth` operation when we found a solution with lower cost than the current cost as a replacement for checking if a solution is in the same homotopy as a previously found solution.

Lazy PRM with Optimization We propose a PRM with lazy evaluations of the edges, and attempt to connect the edges that are not viable using a nonlinear optimizer.



Compared to the `connect` operation, the lazy version of it does not evaluate the edges, and thus necessitates the separate `evaluate` operation. If the whole path is evaluated successfully, and none of the edges are in collision, the process is terminated.

Optimal RRF We propose a probabilistically complete and optimal version of the reconfigurable random forest-algorithm (RRF) proposed in (Li and Shie 2002). RRF is a multiquery-planner that makes use of the proposed data structure and policy to *incrementally* build a roadmap, combining the advantages of single-query, and multi-query planning approaches.



D now being used over several planning queries means that it needs to be *persistent* over the queries to keep track of the state of the graph.

In this algorithm, we explicitly keep track of the connected components in the graph D, and try to extend *each connected component* in the graph towards the previously obtained sample. Initially, this leads to just the start and goal vertices being extended towards the sample. In the following queries, the previous tree(s)/graphs are then also extended towards the samples, leading to quicker initial solutions of the algorithm.

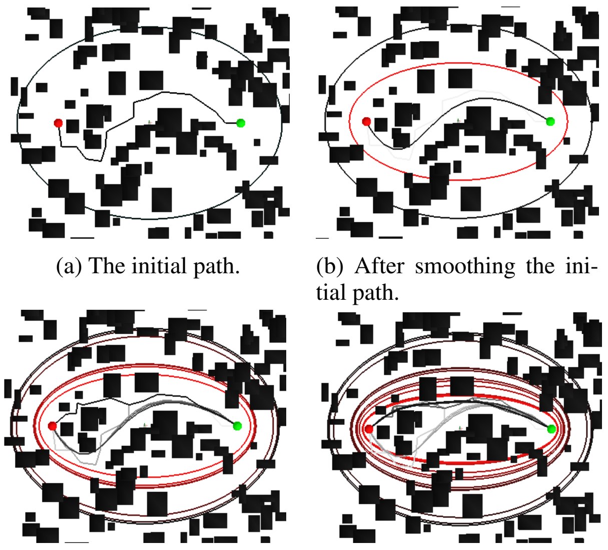
5 Experiments

We run the resulting state machines on the following examples: (i) a cluttered 2D environment, (ii) a 2D problem with a narrow corridor, (iii) a multi-query setting in a cluttered 2D environment (iv) a 12 DoF multi-agent system consisting of a mobile robot and a crane (described in (Hartmann et al. 2020)).

(i) Figure 1 shows snapshots from *Informed RRT* with smoothing* on the cluttered environment. In Figure 2 the evolution of the length of the path is plotted for our algorithm, and Informed RRT* without the smoothing step. In an optimized implementation, the advantage of using a smoother might be smaller, but we argue that the better use of available information is beneficial – especially in higher dimensional problems, where convergence of pure sampling based approaches is slower.

(ii) The evolution of *Lazy PRM with optimization* is shown for the narrow gap environment in Fig. 3. Note that while the example we show here works, we do not expect the optimizer to always find a solution. Hence is might be desirable to first try a `graph search` again after pruning the infeasible edges from the graph. The algorithm might perform very well in cases where the edge is only slightly violating the constraints.

(iii) Figure 4a shows the necessary time required from the RRF to find the solutions to 250 queries on the same environment in comparison to RRT, and Lazy PRM. The number of required collision checks is shown for the planners in Fig. 4b, which shows that many less than RRT, and approximately the same total number as for lazy PRM are necessary.



(a) The initial path.

(b) After smoothing the initial path.

(c) After finding a path in a different topology.

(d) The final path.

Figure 1: Snapshots of the evolution of the Informed RRT* with smoothing. The black path is the lowest-cost-path, the lighter paths are the previously identified solutions. The red ellipsoid is the currently active set from which we sample, the other ellipsoids are the previously used ones.

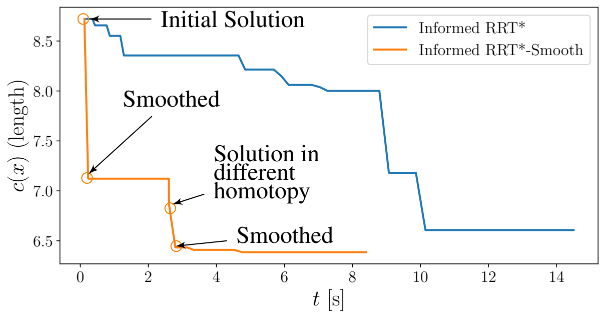
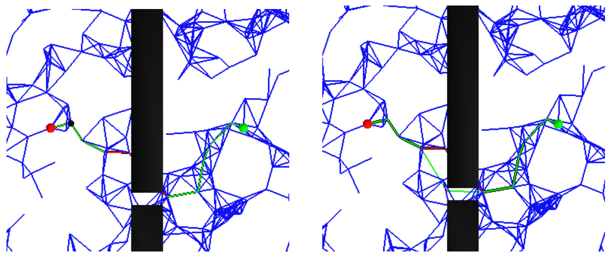


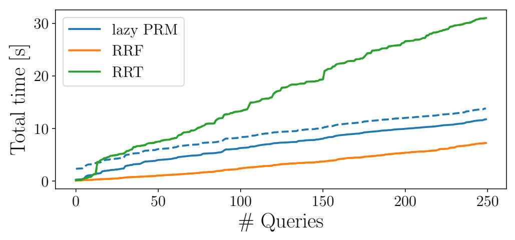
Figure 2: Cost evolution of the *Informed RRT**, and *Informed RRT*-Smooth*-algorithms.



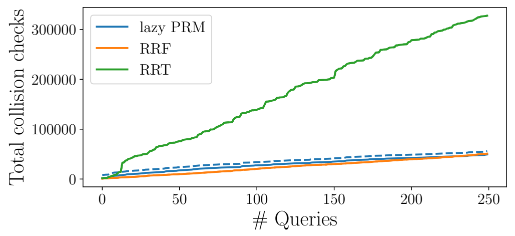
(a) After evaluating the initially found path.

(b) After applying the optimizer to fix the infeasible edge.

Figure 3: Snapshots of the evolution of the lazily evaluated PRM with optimization. Green edges are feasible edges, red edges are not feasible, and blue are not evaluated yet.



(a) Time comparison (dashed lines are preprocessing + online planning time).



(b) Comparison of cumulatively evaluated edges (dashed are the edges checked during preprocessing + online planning).

Figure 4: Results and comparisons for the *Optimal RRF-planner* for 250 queries on the cluttered environment.

(iv) For a more complex example of the *combined robust solver*, see (Hartmann et al. 2020). There, we use the planner to solve a multi agent problem in a cluttered environment. In this use-case, we are more interested in robustness rather than optimality. Detailed statistics of the usage of the optimizer and the RRT-planner are reported to demonstrate the usefulness of the framework.

6 Limitations & Outlook

The presented work is still in the early stages of development. Hence there are still open questions and limitations that need to be investigated in more depth:

- We present the policies implied by known algorithms in our framework, and simple, hand-crafted new policies in this work. Better, more complex ones might be able to highlight the benefits of this framework more clearly.
- The basic operations presented here are still complex and inspired by existing algorithms. This leads to the building blocks being specialized for the use cases we present here. More work is necessary to arrive at a general formulation of the operations that allows to combine them more flexibly. We note that some algorithmic improvements made in path planning research (e.g., sampling a hyper-ellipsoid as in (Gammell, Srinivasa, and Barfoot 2014)) would be hard to identify using this framework.
- There is still a gap in information exchange from sampling-based to optimization based methods. Sampling-based methods, in general, do not incorporate even first order information about the environment.

These limitations lead directly to possibilities for further development:

- Search for a policy π using more advanced methods, e.g., genetic algorithms - this is what we intend as a future goal for this work. The currently presented status is the preliminary work which is necessary to go towards a meta-planning algorithm. We note that learning new algorithms might face typical non-trivial challenges such as sparsity in the solutions domain, and a hard to tune reward function.

7 Conclusion

In this work, we presented a new framework to describe and combine the two different areas of path planning. Using this framework to reason about an algorithm helps make some choices made in common path planners more explicit, and thus improving the algorithms by modifying those choices. Introducing the planning process as a decision process governed by a policy π is the first step towards enabling a policy search that might find new and more performant solvers for the path planning problem. It is thus an initial step towards a meta planner, meaning that we propose a representation for *learning* path planning algorithms.

The examples show that we are able to unify common planners in our framework by using different policies which work with the same underlying data structures and operations. The demonstrations of the algorithm using simple handcrafted policies show that algorithms which are better in some aspects on some planning problems can be described using our framework.

8 Acknowledgements

We thank Joaquim Ortiz de Haro for helpful discussions.

The research has been supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2120/1 – 390831618, and the DAAD.

References

[Branicky et al. 2001] Branicky, M. S.; LaValle, S. M.; Olson, K.; and Libo Yang. 2001. Quasi-randomized path planning. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, volume 2, 1481–1487 vol.2.

[Choudhury et al. 2016] Choudhury, S.; Gammell, J. D.; Barfoot, T. D.; Srinivasa, S. S.; and Scherer, S. 2016. Regionally accelerated batch informed trees (rabit*): A framework to integrate local information into optimal path planning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 4207–4214. IEEE.

[Choudhury et al. 2018] Choudhury, S.; Bhardwaj, M.; Arora, S.; Kapoor, A.; Ranade, G.; Scherer, S.; and Dey, D. 2018. Data-driven planning via imitation learning. *The International Journal of Robotics Research* 37(13-14):1632–1672.

[Gammell, Srinivasa, and Barfoot 2014] Gammell, J. D.; Srinivasa, S. S.; and Barfoot, T. D. 2014. Informed rrt*:

Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2997–3004. IEEE.

[Hartmann et al. 2020] Hartmann, V. N.; Oguz, O. S.; Driess, D.; Toussaint, M.; and Menges, A. 2020. Robust task and motion planning for long-horizon architectural construction planning. In *Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems (IROS)*.

[Herbert et al. 2017] Herbert, S. L.; Chen, M.; Han, S.; Bansal, S.; Fisac, J. F.; and Tomlin, C. J. 2017. Fastrack: A modular framework for fast and guaranteed safe motion planning. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, 1517–1522. IEEE.

[Kalakrishnan et al. 2011] Kalakrishnan, M.; Chitta, S.; Theodorou, E.; Pastor, P.; and Schaal, S. 2011. Stomp: Stochastic trajectory optimization for motion planning. In *2011 IEEE international conference on robotics and automation*, 4569–4574. IEEE.

[Karaman and Frazzoli 2011] Karaman, S., and Frazzoli, E. 2011. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research* 30(7):846–894.

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

[LaValle 1998] LaValle, S. M. 1998. Rapidly-exploring random trees: A new tool for path planning.

[Li and Shie 2002] Li, T.-Y., and Shie, Y.-C. 2002. An incremental learning approach to motion planning with roadmap management. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, volume 4, 3411–3416. IEEE.

[Luna et al. 2013] Luna, R.; Şucan, I. A.; Moll, M.; and Kavraki, L. E. 2013. Anytime solution optimization for sampling-based motion planning. In *2013 IEEE international conference on robotics and automation*, 5068–5074. IEEE.

[Orthey, Escande, and Yoshida 2018] Orthey, A.; Escande, A.; and Yoshida, E. 2018. Quotient-space motion planning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 8089–8096. IEEE.

[Pham, Caron, and Nakamura 2013] Pham, Q.-C.; Caron, S.; and Nakamura, Y. 2013. Kinodynamic planning in the configuration space via admissible velocity propagation. In *Robotics: Science and Systems*, volume 32.

[Ratliff et al. 2009] Ratliff, N.; Zucker, M.; Bagnell, J. A.; and Srinivasa, S. 2009. Chomp: Gradient optimization techniques for efficient motion planning. In *2009 IEEE International Conference on Robotics and Automation*, 489–494. IEEE.

[Schulman et al. 2013] Schulman, J.; Ho, J.; Lee, A. X.; Awwal, I.; Bradlow, H.; and Abbeel, P. 2013. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Robotics: science and systems*, volume 9, 1–10. Citeseer.

[Şucan, Moll, and Kavraki 2012] Şucan, I. A.; Moll, M.; and Kavraki, L. E. 2012. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine* 19(4):72–82. https://ompl.kavrakilab.org.

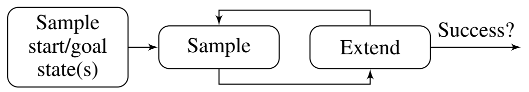
[Toussaint and Lopes 2017] Toussaint, M., and Lopes, M. 2017. Multi-bound tree search for logic-geometric programming in cooperative manipulation domains. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 4044–4051. IEEE.

[Wagner and Choset 2011] Wagner, G., and Choset, H. 2011. M*: A complete multirobot path planning algorithm with performance bounds. In *2011 IEEE/RSJ international conference on intelligent robots and systems*, 3260–3267. IEEE.

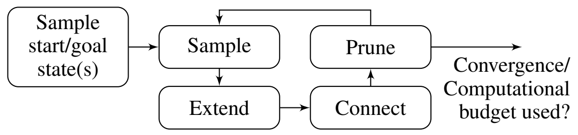
A Known algorithms in our framework

A.1 RRT/RRT*

The following architecture is a typical unidirectional RRT-algorithm. Extensions such as bidirectionality or various mechanisms to bias the growth of the tree(s) can be taken into account by changing the `sample` operation.



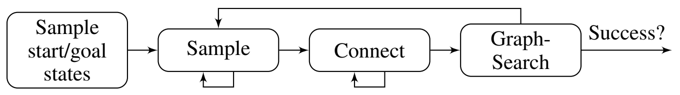
Our version of RRT* takes some elements of a PRM to rewire the tree after the `extend` operation.


Sample start/goal state(s)
Sample
Prune
Extend
Connect
Convergence/ Computational budget used?

RRT* can be readily modified to e.g. Informed-RRT* by using the best current cost to inform the sampling strategy.

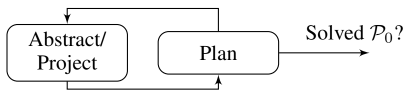
A.2 PRM

The PRM algorithm below is a naive implementation of the known algorithm - no lazy evaluation, or biased/hierarchical sampling is done:



A.3 Quotient space planning

The following architecture can be used to start with a simpler approximation of a complex, high dimensional path planning problem, and inform the later iterations with the previous ones.



Here, we use the `plan`-operation as a meta operation that consists of a full solve of the path planning problem $\mathcal{P}_i$. This can be done via any method described in this paper.