To appear in the Proc. of the IEEE/RSJ Intl. Conf. On Intelligent Robots and Systems (IROS), 2021.

HyperPlan: A Framework for Motion Planning Algorithm Selection and Parameter Optimization

Mark Moll[1], Constantinos Chamzas[1], Zachary Kingston[1], and Lydia E. Kavraki[1]

Abstract— **Over the years, many motion planning algorithms have been proposed. It is often unclear which algorithm might be best suited for a particular class of problems. The problem is compounded by the fact that algorithm performance can be highly dependent on parameter settings. This paper shows that hyperparameter optimization is an effective tool in both algorithm selection and parameter tuning over a given set of motion planning problems. We present different loss functions for optimization that capture different notions of optimality. The approach is evaluated on a broad range of scenes using two different manipulators, a Fetch and a Baxter. We show that optimized planning algorithm performance significantly improves upon baseline performance and generalizes broadly in the sense that performance improvements carry over to problems that are very different from the ones considered during optimization.**

I. Introduction

Producing high-quality motions for complex robots remains a challenging problem. This is in part because the motion planning problem is fundamentally hard (PSPACE-complete, in fact [1]). Nevertheless, a large number of motion planning algorithms have been proposed that perform very well in practice. These algorithms vary not only in their completeness and optimality guarantees, but also in the heuristics used to balance the exploration/exploitation trade-off in their search for a feasible (or optimal) motion plan. There also many other factors that in practice have a huge impact on an algorithm's performance: (a) the extent to which information can be precomputed (and associated computational cost can be amortized), (b) the representation of a robot and its environment (e.g., point clouds vs. meshes vs. collections of spheres), which significantly affect collision and distance calculations, and (c) implementation details. This leads to two closely related problems. For motion planning researchers it becomes difficult to fairly compare against the state of the art, since the performance of any algorithm depends on so many specific details. For practitioners who simply want to use the best algorithm for their specific application it can be an endless trial-and-error process to select an algorithm and tune its parameters to achieve optimal performance. This process is particularly cumbersome for high-dimensional problems where intuition may not help and planning times can be long.

Manuscript received: Month, Day, Year; Revised Month, Day, Year; Accepted Month, Day, Year.

This work was supported in part by NSF 1718478, NSTRF 80NSSC17K0163, and Rice University Funds.

[1]M. Moll, C. Chamzas, Z. Kingston and L.E. Kavraki are with the Department of Computer Science, Rice University, Houston, TX, USA. {mmoll, chamzask, zak, kavraki}@rice.edu

Digital Object Identifier (DOI): see top of this page.

This paper proposes to use a hyperparameter optimization approach to address the problems described above. Hyperparameter optimization has gained in popularity in recent years, in part because it has become an indispensable tool in the deep learning community. The main idea is to model the hyperparameter space, define a loss function that characterizes performance on a particular task, and have an optimization method explore the hyperparameter space so as to minimize the loss value. In the context of motion planning, this optimization can be thought of as choosing the right algorithm and parameter values such that, e.g., total planning time is minimized over a number of representative motion planning problems. The focus in this paper is on a particular class of motion planning algorithms called sampling-based planners [2]. Hyperparameter optimization is especially relevant for this class of algorithms: (1) many such algorithms have been proposed, each with its own parameters, (2) performance of these algorithms can be sensitive to parameter values that are difficult to choose *a priori*, and (3) due to the stochastic nature of such algorithms, characterizing their performance can be non-trivial.

The contributions of this paper are as follows. We present a formulation of motion planning algorithm selection and parameter tuning as a hyperparameter optimization problem. We outline general desiderata for the design of loss functions that capture motion planning algorithm performance and provide several concrete examples. Finally, we extensively evaluate the performance improvements that can be obtained through hyperparameter optimization and characterize the extent that optimized algorithm performance generalizes in realistic settings. While in practice the optimized planner configuration's performance seems to generalize to other planning problems, we suspect that the optimal configuration depends somewhat on a robot's kinematics and the environment geometery. We hope this work will be used in the community to provide a higher standard of algorithm assessment in motion planing.

The rest of the paper is organized as follows. The next section briefly reviews hyperparameter optimization and related work on automatic algorithm configuration, with a focus on robot motion planning. Section III describes the methods used to solve the hyperparameter optimization problem for motion planning. It lays out design considerations for *loss functions* (i.e., optimization objectives for hyperparameter optimization) and includes specific ones that are useful for motion planning such as planning speed or convergence to optimality. Section IV presents an extensive evaluation of the hyperparameter optimization framework for a range of motion planning problems for a mobile manipulator with a variety of loss functions.

II. Related Work

In recent years several approaches to hyperparameter optimization have been proposed. Although this type of optimization can be performed using grid-search or gradient descent techniques [3], stochastic, gradient-free methods [4]–[7] have proven to be popular due to their generality and better scalability compared to alternative methods. While many hyperparameter optimization approaches are focused specifically on deep learning methods, there also several black box approaches that are broadly applicable such Bayesian Optimization [8], Hyperband [9] and their combination (BOHB) [7]. In this paper we use a toolbox called HpBandster [6] that implements these methods, but others exist (e.g., Tune [10]).

There has been some prior work on applying hyperparameter optimization to motion planning [11], [12], but this work focused on the problem of tuning parameters of a given planning algorithm, whereas in this work the planning algorithm is itself another hyperparameter. In [13], the Sequential Model-based Algorithm Configuration (SMAC) [14] is used, a precursor to the HpBandster algorithm used in this work, whereas in [12] a variety of methods is used. Our work generalizes this prior work to a much larger hyperparameter search space (because of the large number of planning algorithms and the large number of algorithm parameters considered), but also discusses the design of effective loss functions and different notions of optimality. In addition, our work provides a more in-depth discussion of how planner configuration optimization generalizes beyond the problems considered during optimization.

III. Methods

A. Introduction

The hyperparameter search space is generally huge in motion planning. Even when restricted to just one library of motion planning algorithms such as the Open Motion Planning Library [15], there are tens of algorithms, each typically having several parameters. Some hyperparameters are categorical, while others are numerical. Additionally, there are often complex dependencies that need to be captured, so that only meaningful combinations of hyperparameter settings are explored. Therefore, a simple brute force search is impractical and a greedy optimization is likely to get stuck in a local minimum. Thus, the problem is to design loss functions that capture relevant aspects of motion planning algorithm performance for use by a more intelligent optimization routine. These loss functions should discriminate—with high probability—higher performing algorithm configurations versus lower performing ones, such that this probability increases as the allowed computational budget increases.

B. Designing Loss Functions

Let the loss L for an algorithm configuration c on a single motion planning problem p be denoted by $L(c,p,t)$. Here, t is a maximum time budget to solve p. For non-deterministic motion planning algorithms (such as sampling-based planners) L is actually a random variable and it is more meaningful to define a loss function over the *distribution* of L. The mean and median could be used, but they do not capture any aspect of the variance of the distribution. In many practical scenarios it is useful to guarantee that planning times remain below some limit with high probability. Also, when given two planning algorithm configurations with the same mean (or median) in their loss distributions, we generally prefer the one with smaller variance so that planning time is more consistent. There is a trade-off, however: fewer runs are necessary to accurately estimate the mean/median than, say, a 99% percentile of the distribution. The latter is useful to bound worst-case performance.

As will become clearer in the next subsection, when minimizing loss functions, it is desirable to have loss functions with the following properties:

- The time spent on evaluating the loss for a planning configuration should be bounded.
- Accuracy of loss estimates is more important for well-performing planning configurations than for poorly-performing planning configurations.
- At the same time, it is helpful if a loss function is highly discriminative, even when planning configurations are poorly performing. In hyperparameter optimization, a large fraction of the search space consists of poorly performing configurations. A loss function that is discriminative is more helpful in directing the search towards more promising areas in the search space.

1) Optimizing for planning speed: Suppose we are interested in finding a motion planning algorithm configuration that is the fastest at reliably solving a set of representative problems $P=\{p_1,\ldots,p_n\}$. Let $m_{t,i}(c)$ denote the number of times that we can solve problem p_i using configuration c within time budget t. The actual solve times $s_j, j=1,\ldots,m_{t,i}$, can be thought as samples drawn from $s_j \sim L(c,p_i,t-\sum_{k=1}^{j-1} s_k)$. Using a slight abuse of notation, we treat these samples as all being drawn from $L(c,p_i,t)$. In many cases (e.g., a small time budget, a difficult planning problem, or poor planning configuration), $m_{t,i}=0$ and thus we have no samples that could help decide which configuration is better than others. In those cases, we can treat the distance $d_{t,i}$ to the goal of the best partial solution path found by a planner configuration as a fallback metric that can be used to help discriminate between different algorithm configurations. We propose to combine these elements in a loss function of this form:

$$L_q(c,p_i,t)=\begin{cases} Q_q(\{s_1,\ldots,s_{m_{t,i}}\}) & m_{t,i}>0,\\ t+d_{t,i}^2 & m_{t,i}=0.\end{cases}$$

Here, $Q_q(\cdot)$ is the q-quantile over a set of samples. With $q=0.5$, $L_q(c,p_i,t)$ estimates the median solve time (if t is large enough). As q is increased the loss function captures more of the expected worst-case performance. In our experiments we used $q=0.7$ to balance capturing worst-case performance with the number of samples required to get a good estimate for the quantile of the underlying distribution from which the samples as drawn[1]. Note the addition of t when $m_{t,i} = 0$ to ensure that the loss is always greater when no solution is found than when one or more solutions are found. The loss over the entire set P is simply defined as the average of individual losses:

$$L_q(c,P,t) = \tfrac{1}{n}\sum_{i=1}^{n} L_q(c,p_i,t)$$

It is straightforward to verify that this function has the three desirable properties stated above:

- The time to compute $L_q(c,P,t)$ is bounded and roughly equal to $n \cdot t$ (there is some additional computational overhead in initializing a motion planning problem).
- For a sufficiently large time budget, a good planner configuration is likely to solve a given problem more often than a bad one, leading to a more accurate estimate of the loss for a good configuration.
- It is discriminative: for a small time budget (where a planner configuration may not be able to solve a problem), a good configuration is more likely to get closer to the goal than a bad one. Note that if $m_{t,i} > 0$, $Q_q(\{s_1,\ldots,s_{m_{t,i}}\}) \le t$, which means that the loss for configurations with $m_{t,i} = 0$ is always strictly larger than the loss for configurations with $m_{t,i} > 0$.

2) Optimizing combined planning time and execution time: In online planning, motion planning and execution are typically interleaved: while a robot is executing a plan, it is already planning a new path starting from an (expected) future state. This is often done at a certain frequency. Obviously, the planner needs to be fast enough that a trajectory is found before the expected start state is reached. At the same time, a shorter trajectory length is preferable, so a planner configuration that is slightly slower than others, but produces very short trajectories could be considered more optimal. A way to optimize both is to consider optimization of the *combined* planning time and execution time, where execution time is simply the duration of the trajectory produced by the planner. Let the loss of a single run be defined as

$$D_j(c,p_i,t) = \begin{cases} s_i + \ell_i & s_i \le t, \\ t + \ell_{\max} + d_{t,i}^2 & \text{otherwise,} \end{cases}$$

Here, s_i is the time the planner configuration c took to solve problem p_i and produce a trajectory with duration ℓ_i, if it could be solved within the time limit t. If the problem could not be solved, the loss is equal to the time limit plus $\ell_{\max}$, a user-defined upper bound on path length, plus the square of the distance to the goal for the best approximate solution found. The loss functions $D_q(c,p_i,t)$ and $D_q(c,P,t)$ can again be defined analogous to $L_q(\cdot)$:

$$D_q(c,p_i,t) = Q_q(\{D_j(c,p_i,t) | j = 1,\ldots,m_{t,i}\}),$$
$$D_q(c,P,t) = \frac{1}{n}\sum_{i=1}^{n} D_q(c,p_i,t).$$

By inspection we can verify that the loss function $D_q(c,P,t)$ also has the desirable properties stated at the start of the section.

3) Optimizing for convergence to optimality: In some applications it is not only important to quickly find a feasible solution, but ideally find one that is close to optimal. Many sampling-based planning algorithms have been proposed that guarantee asymptotic (near-)optimality (see, e.g., [17]–[19]): as time approaches infinity the solution path converges on the optimal path (or a constant factor approximation thereof in the case of near-optimality). Many such planners can be run as anytime algorithms: they can return the best found solution at any given time. The ideal anytime planner would quickly find a feasible solution and thereafter quickly converge to the optimal solution. Suppose t_0 is the time at which a particular planner configuration finds an initial feasible path. Let $\ell(t), t \ge t_0$ be the cost of the path at time t. The cost of a solution path can be defined in various ways, but for simplicity it is often assumed to be path length. For asymptotically optimal planners we can now define the loss for a single run j of a planner configuration as the area under the curve of path cost as a function of time:

$$C_j(c,p_i,t) = \begin{cases} \frac{1}{t}\left(t_0 \cdot \ell(t_0) + \int_{\tau=t_0}^{t} \ell(\tau)\right) d\tau & t_0 \le t, \\ \ell_{\max} + d_{t,i}^2 & d_{t,i} \text{ is finite} \\ 2\ell_{\max} & \text{otherwise.} \end{cases}$$

Here, $\ell_{\max}$ is a user-defined maximum cost constant to be used when no solution was found within time t seconds and $d_{t,i}$ denotes distance to the goal as before. Note that asymptotically optimal planners typically do not terminate until the maximum solve time is exceeded: they keep running, trying to find a better solution. We therefore use the time budget t for m_t runs of exactly $\frac{t}{m_t}$ seconds each. The number of runs m_t can be a function of t, but in our experiments we simply fixed it to a constant equal to 10. The loss functions $C_q(c,p_i,t)$ and $C_q(c,P,t)$ can now be defined analogous to $L_q(\cdot)$:

$$C_q(c,p_i,t) = Q_q(\{C_j(c,p_i,t) | j = 1,\ldots,m_t\}),$$
$$C_q(c,P,t) = \tfrac{1}{n}\sum_{i=1}^{n} C_q(c,p_i,t).$$

By inspection we can verify that the loss function $C_q(c,P,t)$ also has the desirable properties stated at the start of the section.

C. *Optimizing Loss Functions*

In this work, we use a framework for hyperparameter optimization called Bayesian Optimization and Hyperband (BOHB) [7], although other frameworks could be used as well. As the name suggests, BOHB combines the strengths of two previously proposed hyperparameter optimization techniques:

[1]The variance of $Q_q(\cdot)$ is equal to $\frac{q(1-q)}{m_{t,i} f(t)}$, where $f(t)$ is the probability density of the (unknown) true distribution of solve times [16].

Bayesian Optimization [8] and Hyperband [9]. Hyperband optimizes hyperparameters by evaluating configurations in a sequence of stages with budgets that increase by a scaling factor η with each successive stage (the default value of $\eta = 3$ was used in this paper). The best $\frac{1}{\eta}$ configurations of each stage are then advanced to the next stage. The total number of configurations considered at each stage is reduced by a factor of 2 from the previous stage. The initial set of configurations and a $(\frac{1}{2} - \frac{1}{\eta})$ fraction of configurations in subsequent stages are randomly sampled. BOHB improves on this basic scheme by replacing random sampling in each stage with Bayesian optimization. The method has been shown to have strong anytime and final performance [7]. Moreover, it lends itself well to a parallel implementation: the loss values for many different hyperparameter settings can be evaluated independently. The particular implementation of BOHB we used, HpBandster[2], supports different types of hyperparameters (e.g., categorical, integer, and continuous) as well as constraints between them (e.g., certain hyperparameters might only be relevant for specific values of other hyperparameters). All of these features are essential for the problem of motion planning algorithm selection and parameter tuning. For example, the motion algorithm type is a categorical variable and a parameter for the number of nearest neighbors to connect to is an integer variable, but that parameter is only relevant for planners like PRM.

D. Greedy Planning Portfolio Selection

A by-product of running BOHB is that we get a diverse set of planning configurations that (at least in the last stages of Hyperband) tend to perform quite well. Additionally, we can also easily log the loss value for each of these planning configurations on *individual* planning problems (in addition to the aggregate loss value). This can be useful to synthesize a portfolio of planning configurations that has the potential to outperform the best one found. Suppose we run *n* planning threads in parallel, all solving the same problem and terminating as soon as one finds a solution. This has been shown to give superlinear speedups over a single planning thread when using randomized algorithms [20]–[22]. However, the best configuration overall may not necessarily be the best planner for every individual planning problem. Creating a portfolio of algorithms to be run in parallel is complementary to the question of how any individual algorithm can be effectively parallelized. There are some fundamental differences between these problems, however. With a portfolio of randomized algorithms, randomness is exploited to achieve speedups rather than explicit coordination among threads. Furthermore, the portfolio approach essentially selects and tunes a set of search heuristics that optimizes performance across some set of problems.

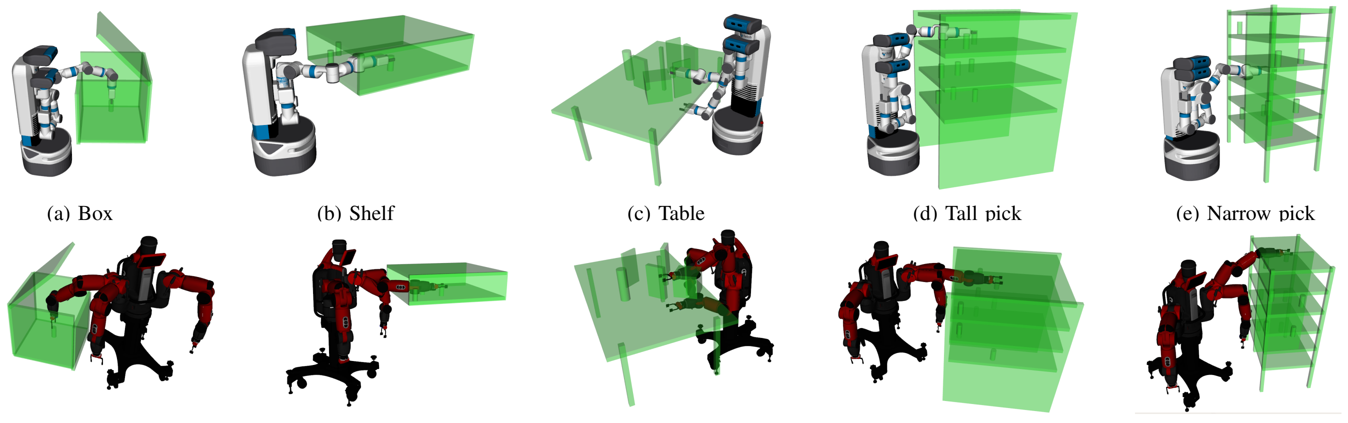
Note that when combining randomized algorithms into a portfolio, a higher variance in run times is not necessarily a problem. In fact, it can actually be an advantage. Suppose we have two algorithm configurations with identical mean and median in their run times, but significantly different variances. Then a portfolio with *n* configurations of the higher variance configuration will likely outperform one with *n* configurations of the lower variance configuration, since the probability that at least 1 of the *n* threads finds a solution in a time that is significantly below the mean/median is much higher in the former case. Of course, this advantage needs to be balanced with bounding the worst-case behavior and depends both on the exact distribution of solve times and the number of threads available. By using a quantile threshold that is significantly below 0.5 in the loss functions defined above, the hyperparameter optimization will effectively optimize configurations for *best*-case performance, which may be useful in generating configurations for high-performing portfolios of planning algorithms run in parallel. More sophisticated schemes for running multiple algorithms in parallel have been proposed (e.g., using time allocation based on multi-armed bandits [23]). Investigating portfolio selection and effectively leveraging such a portfolio is left for future work.

IV. Evaluation

Experimental setup: The HyperPlan framework has been implemented in an extensible way. The results in this section were obtained using HpBandSter as the hyperparameter optimization framework. The algorithms included for optimization are all available in the Open Motion Planning Library (OMPL) [15]. A generic helper program benchmarks a particular planner configuration on a particular motion planning problem and returns a log file of performance characteristics. This output is parsed by HyperPlan and aggregated into a loss value. The helper program relies on OMPL's benchmarking capabilities [24] and MoveIt [25], which makes it possible to benchmark algorithms in realistic settings. Other helper programs for different robotics software frameworks can easily be added.

Fig. 1 shows a number of test cases used to evaluate HyperPlan. In each case, the motion planning problem is to find a collision-free path between two joint states of manipulator. In the top row (Fig. 1(a)–(e)), test cases are based on planning for the 7-DOF arm and 1-DOF torso of a Fetch mobile manipulator, while in the bottom row (Fig. 1(f)–(j)) a 7-DOF arm of a Baxter manipulator is used. Each image shows the start and goal state of the robot superimposed. The start state is either a configuration in a folded "home" position or a configuration where the gripper is at a lower height than the goal. The base remains in the same place. Each example shown is actually representative of a *class* of problems. The base pose of the robot as well as the exact positions of the objects on the shelves and table are uniformly randomly sampled (see [26] for details). For each class we generated 20 different motion planning problems this way. Hyperparameter optimization was applied to the first three problems of each class, while the remaining 17 problems and all other motion planning problems from other classes were used to evaluate how performance generalizes. We also conducted an experiment where HyperPlan was run using one motion planning problem each from the Box, Shelf, and Table

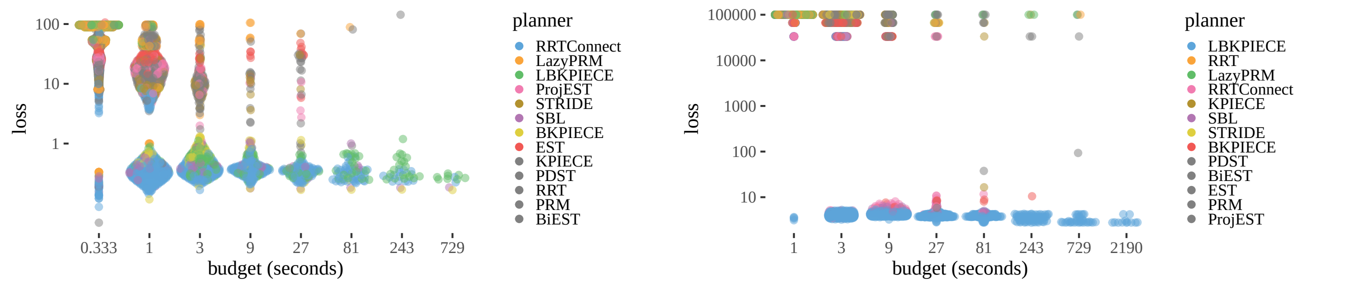
[2]See https://automl.github.io/HpBandSter.



(a) Box (b) Shelf (c) Table (d) Tall pick (e) Narrow pick

(f) Box (g) Shelf (h) Table (i) Tall pick (j) Narrow pick

Fig. 1. Test cases used for evaluating HyperPlan.



(a) Box test case, L_q loss function (b) Box/Shelf/Table, D_q loss function

Fig. 2. Visualization of the hyperparameter search space exploration for two test cases. Lower loss values are better. **(a)** Aggregate performance across three problems instances of the Box class where planning speed is optimized for the Fetch. **(b)** Aggregate performance across one problem each from the Box, Shelf, and Table classes where combined planning and execution time is optimized for the Fetch.

classes. This is useful to evaluate whether a more diverse set of motion planning problems as input to HyperPlan also results in better generalization behavior on other problems.

As a baseline for performance we use the default planner configuration used in the Fetch MoveIt configuration package. This happens to be RRTConnect [27], with parameters heuristically selected by OMPL. The hyperparameter search space explored during optimization includes many planning algorithms and many parameters; a detailed description can be found in the appendix.

Visualization of hyperparameter space exploration: Fig. 2 shows a visualization of the hyperparameter optimization process for two different test cases. On the X-axis the different rounds of hyperband optimization are shown with the corresponding budget in seconds. Each dot represents a planner configuration sampled by BOHB. The Y-axis shows the corresponding loss value on a log scale. The dots are colored by planner type and the planners are ordered in the legend by how often they were sampled. Planner configurations with the fewest samples are all colored in gray. The number of sampled configurations decreases exponentially with increasing budget size. Initial loss values tend to vary significantly in part because HpBandSter does not have a good model yet, but also because the loss values themselves are coarse estimates of loss values for the same planner configurations at higher budgets. In Fig. 2(a) HyperPlan quickly discovers many configurations of RRTConnect with low loss values, but a single configuration each for SBL and BKPIECE maintain a slight advantage over RRTConnect in the last rounds of optimization. In Fig. 2(b) LBKPIECE is found to be optimal from the beginning. The optimization process will still sample other types of planners, but almost all of them perform significantly worse.

Results for optimizing planning speed and combined planning and execution time: Table I summarizes the results from a large suite of experiments. Each row corresponds to the environments shown in Fig. 1. The first two groups of rows show results for the Fetch and Baxter, respectively, when optimizing for planning speed. The last two groups of rows show results for the Fetch and Baxter, respectively, when optimizing for combined planning and execution time. The optimized planner configuration is described in the second column and the corresponding loss value is shown in the next column. This loss value can interpreted as follows: in the “Box” test case in the very first row, any of of three sample problems used during optimization can be solved within 0.17 seconds 70% of the time (since a quantile of 0.7 was used). In contrast, the baseline performance for that same setting is almost twice as large at 0.27 seconds.

TABLE I

HyperPlan results for Fetch and Baxter when optimizing for L_q (planning speed) and D_q (combined planning and execution speed) across a range of problem classes. All entries correspond to loss values for different planner configurations evaluated on different sets of motion planning problems.

	problem class	optimized planner configuration	optimized	baseline	octomap	baseline, octomap	within class	across classes
Fetch, L_q	Box	BKPIECE, shoulder proj., range=1.17	0.17	0.27	0.30	1.51	0.12	2.51
	Shelf	LBKPIECE, shoulder proj., range=2.01	1.21	51.21	43.03	282.70	2.51	2.25
	Table	LBKPIECE, shoulder proj., range=1.02	0.07	0.82	0.35	24.00	0.09	0.53
	Tall pick	LBKPIECE, shoulder proj., range=1.28	1.57	64.14	6.06	353.24	2.36	0.79
	Narrow pick	LBKPIECE, shoulder proj., range=327	11.91	495.04	356.29	572.84	9.84	7.20
	Box/shelf/table	LBKPIECE, shoulder proj., range=1.03	0.13	3.19	0.57	33.67	0.29	0.70
Baxter, L_q	Box	RRTconnect, no intermediate states, range=2.35	0.10	0.15	0.22	0.37	0.12	18.82
	Shelf	LBKPIECE, shoulder projection, range=341	4.99	52.47	33.04	334.98	4.59	16.18
	Table	RRTconnect, keep intermediate states, range=1.93	0.25	0.95	4.28	245.52	0.28	26.42
	Tall pick	RRTconnect, no intermediate states, range=2.45	2.70	18.85	8.69	153.25	3.70	22.14
	Narrow pick	RRTconnect, no intermediate states, range=344	N/A	512.55	487.74	487.87	598.67	142.15
	Box/shelf/table	LBKPIECE, shoulder proj., range=1.18	0.38	7.87	0.88	80.40	0.38	2.00
Fetch, D_q	Box	SBL, shoulder proj., range=1.02	2.83	3.86	3.41	5.98	2.78	3.41
	Shelf	LBKPIECE, shoulder proj., range=1.37	3.57	76.30	10.99	N/A	3.79	3.81
	Table	LBKPIECE, shoulder proj., range=1.14	2.55	3.79	3.50	27.82	2.63	3.68
	Tall pick	LBKPIECE, shoulder proj., range=260	26.73	55.33	87.62	N/A	27.79	10.40
	Narrow pick	LBKPIECE, shoulder proj., range=373	20.11	137.45	103.05	N/A	12.23	13.39
	Box/shelf/table	LBKPIECE, shoulder proj., range=1.06	2.76	6.12	3.75	40.46	2.98	3.54
Baxter, D_q	Box	RRTConnect, keep intermediate states, range=1.57	1.47	1.81	1.96	2.33	1.74	N/A
	Shelf	LBKPIECE, shoulder proj., range=401	7.93	75.03	35.05	N/A	6.80	N/A
	Table	RRTConnect, no intermediate states, range=3.52	2.66	3.27	15.40	122.69	2.75	N/A
	Tall pick	LBKPIECE, shoulder proj., range=3.81	5.10	25.06	12.11	149.65	6.58	N/A
	Narrow pick	LBKPIECE, shoulder proj., range=29.6	68.88	N/A	45.44	N/A	N/A	18.51
	Box/shelf/table	LBKPIECE, shoulder proj., range=1.36	2.52	6.36	3.16	52.44	2.64	7.57

The remaining columns characterize generalization. First, for each environment we generated an artificial octomap [28] of each environment and evaluated the optimized planner configuration on this representation. (Octomaps are a common environment representation created by a perception pipeline.) The loss value for this setting is shown in the "octomap" column. We can also evaluate the loss value for the baseline on the same octomap examples, shown in the "baseline, octomap" column. The octomap representation is more complex than the original mesh representation, which tends to increase the cost of collision checking, often the dominant computational cost in motion planning. The octomap representation also slightly inflates the obstacle size, which makes the motion planning problems even harder. As can be seen in Table I, the planner configuration that was optimized for simple mesh environments also offers significant performance improvements when the environment is represented by an octomap. The "within class" column shows the loss value when the optimized planner configuration is used on the remaining 17 sample problems from that class. Finally, the last column shows the loss value when using all motion problems except the ones during hyperparameter optimization. Loss values greater than 600 are listed as "N/A," since they correspond to cases where the planner configuration found by HyperPlan timed out on one or more of the test cases (in which case the loss values are not very meaningful). The results in the last column suggest that in general (across robots and loss functions) optimizing performance using a diverse set of motion planning problems (i.e., the "box/shelf/table" rows), results in planning configurations with (near-)best overall performance across all classes. Furthermore, performance improvements generally transfer well to the octomap representation. Optimizing over simpler problems (such as the "box" and "table" test cases) typically also results in good generalization behavior, but this is not as robust.

Results for optimizing for convergence: In a second experiment, the convergence to optimality is optimized using three instances of the Box class for the Fetch manipulator. HyperPlan arrive at AIT* with 1347 samples per batch and using k-nearest neighbors (rather than r-nearest neighbors) as the best-performing planning configuration with the following losses:

optimized	baseline	octomap	baseline, octomap	w/in class
2.01	N/A	2.26	N/A	2.30

Note that loss values do not represent time but area under a curve, and are thus harder to interpret. Nevertheless, performance on the three examples used for optimization seems to generalize to the other 17 examples. Surprisingly, performance on the corresponding octomap representations for the three scenes used during optimization is only marginally worse. Performance across other problem classes is not reported, because the optimized configurations were not able to solve many problem instances from other classes. Note that for this loss function is not clear what algorithm should be used for baseline performance. RRT*, one of the first asymptotically optimal algorithms, would a be reasonable choice, except that with default parameter settings it is not able to solve many problem instances within five minutes.

V. Conclusion

We have presented an approach to cast motion planning algorithm selection and parameter tuning as a hyperparameter optimization problem. This is a difficult problem since there are many algorithms to choose from, and the best one for a particular class of problems is highly dependent on those problems, the particular parameter settings, and the notion of optimality used. The focus in this paper has been on sampling-based planners, where this problem of algorithm selection is further complicated by the stochastic nature of these algorithms. We presented general guidelines for constructing loss functions to characterize the performance of planning algorithm configurations and present results for two different manipulators on a variety of problems. We showed that optimized planner configurations generally perform well on other problem classes not considered during optimization.

The results in this paper indicate that performance improvements of optimized planner configurations generalize broadly. This suggests that a way to bootstrap optimization over hard, time-consuming motion planning problems is to seed the initial set of planner configurations with ones optimized for simpler problems that require only a small time budget. In future work, we plan to explore this type of curriculum learning to accelerate hyperparameter optimization. We also plan to further investigate how the output from hyperparameter optimization can be used for algorithm portfolio selection and how a such portfolio can be used effectively during online planning.

Appendix

A. *Optimizing Planning Speed*

The hyperparameter configuration space consisted of the following sampling-based planning algorithms: PRM [29], Lazy PRM [30], RRT [27], RRTConnect [27], EST [31], bidirectional EST, KPIECE [32] (including bidirectional and lazy versions), SBL [33], and STRIDE [34]. The following planner parameters were also included:

- **Max. number of nearest neighbors:** This determines the number of states that PRM and LazyPRM attempt to connect to any new valid sampled state. It is an integer value in the range $[1,20]$.
- **Goal bias:** This controls how aggressively exploration is biased towards expanding a roadmap/tree directly toward a goal state. This is a real-valued parameter in the range $[0,1]$.
- **Range:** This controls the maximum distance that a tree is extended before creating a new state. This is a real-valued parameter in the range $[1,500]$.
- **Intermediate states:** This boolean flag controls whether to keep intermediate states that are checked for collisions during the extension of a tree.
- **Type of projection:** Low-dimensional projections are often useful for kinematic chains and other high-dimensional systems to keep track of search space coverage. Many planners [31]–[33] rely on such projections to guide the exploration. While infinitely many projections could be defined, we restricted the projection to two possible options: (1) a projection onto the first two joint angles (as the joints closest to the base tend to cause the largest displacement of the arm) and (2) a projection onto the end effector position, as computed by the forward kinematics.

These parameters are not applicable to all planning algorithms. HpBandster includes the ability to capture dependencies between hyperparameters so that, e.g., the maximum number of neighbors is only used for PRM and Lazy PRM and the intermediate states are only used for RRT and RRTConnect.

B. *Optimizing Combined Planning and Execution Speed*

Other than the loss function, the experimental setup is the same as in the optimization for planning speed.

C. *Optimizing for Convergence to Optimality*

The hyperparameter configuration space consisted of the following sampling-based planning algorithms: RRT* [17], RRT^X [35], BIT* [19], AIT* [36], LBTRRT [37], and SST [38]. Note that we cannot use the same planning algorithms as in the previous example, since those do not provide any optimality guarantees. The following planner parameters were also included:

- **Goal bias:** See definition above.
- **Range:** See definition above.
- **Whether to use k-nearest or r-nearest neighbors:** This boolean parameter determines which variant is used for selecting the optimal neighborhood for connecting to nearby states and rewiring a tree.
- **Samples per batch:** This is an integer parameter used by BIT* and AIT*. Values were sampled on a logarithmic scale over the range $[1,10000]$.
- **Delay collision checking:** A boolean flag for RRT* that delays collision checking between a state and its neighbors.
- **Rejection variant:** A categorical variable that selects one of four variants of a subroutine for RRT^X.
- **Selection and pruning radii:** Two parameters for SST, sampled over the range $[.01,5]$.

References

[1] J. Canny, *The Complexity of Robot Motion Planning*. Cambridge, MA: MIT Press, 1988.

[2] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *IEEE Access*, vol. 2, pp. 56–77, 2014.

[3] D. Maclaurin, D. Duvenaud, and R. Adams, "Gradient-based hyperparameter optimization through reversible learning," in *International Conference on Machine Learning*, 2015, pp. 2113–2122.

[4] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan *et al.*, "Population based training of neural networks," *arXiv preprint arXiv:1711.09846*, 2017.

[5] J. Lorraine and D. Duvenaud, "Stochastic hyperparameter optimization through hypernetworks," *arXiv preprint arXiv:1802.09419*, 2018.

[6] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, J. Marben, P. Müller, and F. Hutter, "BOAH: A tool suite for multi-fidelity bayesian optimization & analysis of hyperparameters," *arXiv preprint arXiv:1908.06756*, 2019.

[7] S. Falkner, A. Klein, and F. Hutter, "BOHB: Robust and efficient hyperparameter optimization at scale," in *Intl. Conf. on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 1437–1446.

[8] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, vol. 25, pp. 2951–2959, 2012.

[9] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6765–6816, 2017.

[10] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, "Tune: A research platform for distributed model selection and training," *arXiv preprint arXiv:1807.05118*, 2018.

[11] R. Burger, M. Bharatheesha, M. van Eert, and R. Babuška, "Automated tuning and configuration of path planning algorithms," in *IEEE Intl. Conf. on Robotics and Automation*, May 2017, pp. 4371–4376.

[12] J. Cano, Y. Yang, B. Bodin, V. Nagarajan, and M. O'Boyle, "Automatic parameter tuning of motion planning algorithms," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, Oct. 2018, pp. 8103–8109.

[13] M. Streeter and S. F. Smith, "New techniques for algorithm portfolio design," *arXiv preprint arXiv:1206.3286*, 2012.

[14] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Intl. conf. on learning and intelligent optimization*. Springer, 2011, pp. 507–523.

[15] 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, Dec. 2012, http://ompl.kavrakilab.org.

[16] A. Stuart and K. Ord, *Kendall's Advanced Theory of Statistics*. London: Arnold, 1994.

[17] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Intl. J. of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[18] J. D. Marble and K. E. Bekris, "Asymptotically near-optimal planning with probabilistic roadmap spanners," *IEEE Trans. on Robotics*, vol. 29, no. 2, pp. 432–444, 2013.

[19] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa, "Batch informed trees (BIT*): Informed asymptotically optimal anytime search," *Intl. J. of Robotics Research*, vol. 39, no. 5, pp. 543–567, 2020.

[20] V. Rao and V. Kumar, "Superlinear speedup in parallel state-space search," in *Foundations of Software Technology and Theoretical Computer Science*, ser. Lecture Notes in Computer Science, K. Nori and S. Kumar, Eds. Springer Berlin / Heidelberg, 1988, vol. 338, pp. 161–174.

[21] J. Ichnowski and R. Alterovitz, "Parallel sampling-based motion planning with superlinear speedup," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, Oct. 2012, pp. 1206–1212.

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

[22] M. Otte and N. Correll, "C-FOREST: Parallel shortest path planning with superlinear speedup," *IEEE Trans. on Robotics*, vol. 29, no. 3, pp. 798–806, June 2013.

[23] M. Gagliolo and J. Schmidhuber, "Algorithm portfolio selection as a bandit problem with unbounded losses," *Annals of Mathematics and Artificial Intelligence*, vol. 61, no. 2, pp. 49–86, 2011.

[24] M. Moll, I. A. Şucan, and L. E. Kavraki, "Benchmarking motion planning algorithms: An extensible infrastructure for analysis and visualization," *IEEE Robotics & Automation Magazine (Special Issue on Replicable and Measurable Robotics Research)*, vol. 22, no. 3, pp. 96–102, Sep. 2015.

[25] I. A. Şucan and S. Chitta, "MoveIt!" http://moveit.ros.org.

[26] C. Chamzas, Z. Kingston, C. Quintero-Peña, A. Shrivastava, and L. E. Kavraki, "Learning sampling distributions using local 3d workspace decompositions for motion planning in high dimensions," *arXiv preprint arXiv:2010.15335*, 2020.

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

[28] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, 2013. [Online]. Available: http://octomap.github.com

[30] R. Bohlin and L. E. Kavraki, "Path planning using lazy PRM," in *Proc. 2000 IEEE Intl. Conf. on Robotics and Automation*, San Francisco, CA, 2000, pp. 521–528.

[31] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," *Intl. J. of Computational Geometry and Applications*, vol. 9, no. 4-5, pp. 495–512, 1999.

[32] I. A. Şucan and L. E. Kavraki, "A sampling-based tree planner for systems with complex dynamics," *IEEE Trans. on Robotics*, vol. 28, no. 1, pp. 116–131, 2012.

[33] G. Sánchez and J.-C. Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking," in *The Tenth International Symposium on Robotics Research*, 2001, pp. 403–417.

[34] B. Gipson, M. Moll, and L. E. Kavraki, "Resolution independent density estimation for motion planning in high-dimensional spaces," in *IEEE Intl. Conf. on Robotics and Automation*, 2013, pp. 2429–2435.

[35] M. Otte and E. Frazzoli, "RRT$^{\mathrm{X}}$: Real-time motion planning/replanning for environments with unpredictable obstacles," in *Algorithmic Foundations of Robotics XI*. Springer, 2015, pp. 461–478.

[36] M. P. Strub and J. D. Gammell, "Adaptively informed trees (ait*): Fast asymptotically optimal path planning through adaptive heuristics," in *icra*, May 2020, pp. 3191–3198.

[37] O. Salzman and D. Halperin, "Asymptotically near-optimal rrt for fast, high-quality motion planning," *IEEE Trans. on Robotics*, vol. 32, no. 3, pp. 473–483, June 2016.

[38] Y. Li, Z. Littlefield, and K. E. Bekris, "Asymptotically optimal sampling-based kinodynamic planning," *Intl. J. of Robotics Research*, vol. 35, pp. 528–564, Apr. 2016.