Automatic Parameter Tuning of Motion Planning Algorithms

José Cano, Yiming Yang, Bruno Bodin, Vijay Nagarajan, and Michael O'Boyle
School of Informatics, University of Edinburgh, UK

Abstract— **Motion planning algorithms attempt to find a good compromise between planning time and quality of solution. Due to their heuristic nature, they are typically configured with several parameters. In this paper we demonstrate that, in many scenarios, the widely used default parameter values are not ideal. However, finding the best parameters to optimise some metric(s) is not trivial because the size of the parameter space can be large. We evaluate and compare the efficiency of four different methods (i.e. random sampling, AUC-Bandit, random forest, and bayesian optimisation) to tune the parameters of two motion planning algorithms, BKPIECE and RRT-connect. We present a table-top-reaching scenario where the seven degrees-of-freedom KUKA LWR robotic arm has to move from an initial to a goal pose in the presence of several objects in the environment. We show that the best methods for BKPIECE (AUC-Bandit) and RRT-Connect (random forest) improve the performance by 4.5x and 1.26x on average respectively. Then, we generate a set of random scenarios of increasing complexity, and we observe that optimal parameters found in simple environments perform well in more complex scenarios. Finally, we find that the time required to evaluate parameter configurations can be reduced by more than 2/3 with low error. Overall, our results demonstrate that for a variety of motion planning problems it is possible to find solutions that significantly improve the performance over default configurations while requiring very reasonable computation times.**

I. INTRODUCTION

Motion planning is the fundamental problem of finding a valid path from a start to goal pose [1]. In particular, a large number of sampling-based motion planning algorithms have been developed during the past few decades, such as Rapidly-exploring Random Tree (RRT, [2]), Probabilistic Roadmap (PRM, [3]), and many others [4]. Sampling-based methods are often used for efficiently finding globally valid solutions in complex environments. Thus, a common demand from the users when deploying motion planning to solve real problems is that they want to have a valid solution as quick as possible, i.e., with minimum planning time. However, the planning time can be significantly affected by the set of parameters associated with a particular planning algorithm. Typically, the users (especially non-expert ones) adopt the default parameter values provided by the algorithm designers, since these default configurations may be reasonable settings for the *average* case, although their behavior may be far from optimal (minimum planning time) for any *specific* planning problem. In this paper, we demonstrate that default configurations give sub-optimal performance and that there exists other configurations that provide significant improvement in particular cases. Note that we focus on the performance of the application (in our case the planning time) and not on the system, so we don't consider energy efficiency.

However, for some algorithms the high number of parameter combinations is translated into a large search space. Finding the best parameter configuration within this search space is tedious and time consuming. As an example, we propose two categories of environments where a seven Degrees-of-Freedom (DoF) KUKA LWR robot arm has to find a collision-free trajectory to move from an initial to a goal pose in the presence of several collision objects, using two different planning algorithms, the Bidirectional RRT (RRT-Connect, [5]) and the Bidirectional Kinodynamic motion planning by Interior-Exterior Cell Exploration (BKPIECE, [6]). BKPIECE can be configured through five parameters that generate a search space of $1,760,000$ possible combinations, and fully exploring it to find optimal solutions may require weeks. Thus, a tool that can automatically find the optimal parameters for particular planning algorithms and/or environments in short times is of interest to the robotics community and the main target of this work.

There have been some previous works addressing parameter configuration of motion planning algorithms. In [7], there are only two parameters to configure that generate a very small search space, thus they are explored manually. Luo and Hauser [8] compare the performance of various planning algorithms by tuning the parameters of some of them to "stable values". The number of parameters tuned is at most three, but the tuning process itself was not specified. The closest related work in the literature is [9], where the SMAC tool was proposed to tune the parameters of several motion planners on specific problems. However, this work evaluates only one tuning method (SMAC which is based on random forest), targets less complex environments than us and does not perform robustness and randomness experiments, as we do. In [10] it is proposed a benchmarking infrastructure to compare the performance of different planners, however nothing is said about tuning the individual parameters of each algorithm. Finally, [11], [12], [13] address the configuration problem within a distributed system where several processes/algorithms can be tuned with only one parameter. However, the parameter values can be affected by the specific allocations of processes to processors within the system.

In this paper we evaluate and compare four parameter tuning methods: 1) simple random sampling; 2) the AUC-Bandit approach; 3) bayesian optimisation; and 4) random forest. It is important noting that the time required for each method to find the optimal parameters is also critical. The users may not be interested in finding the optimal parameters in hours (or days) while the actual reduction in planning time is only a few seconds or less. We limit this time to two hours.

The two main contributions of the paper are as follows:

- We find that for specific real-world motion planning problems table-top-reaching, it is possible to find parameters that make the motion planner 4.5x faster than using the default parameters under two hours.
- We show that optimal parameters found in simple environments perform well also in more complex scenarios. This is a critical discovery, meaning that one can train the system quickly in simple environments while being able to use the parameters in complex scenarios.

II. PROBLEM DEFINITION

Let $\mathbf{q} \in \mathcal{C} \subset \mathbb{R}^N$ denote the state of a N Degrees-of-Freedom (DoF) robot, where $\mathcal{C}$ is the configuration space. In an environment Env, let $\mathcal{C}_{obs}$ denote the manifold on which the robot is in collision with environmental obstacles or itself, and $\mathcal{C}_{free} = \mathcal{C} \backslash \mathcal{C}_{obs}$ the manifold of collision-free states. Given a start and goal state, $\mathbf{q}_s$ and $\mathbf{q}_g$, a motion planning problem is defined as:

$$\begin{array}{c} \mathbf{q}_{[s:g]} = MotionPlan(\mathbf{q}_s, \mathbf{q}_g) \\ s.t. \quad \mathbf{q}_{[s:g]} \subset \mathcal{C}_{free} \end{array}, \tag{1}$$

Let A denote a motion planning algorithm that can solve (1). A can be configured by a set of parameters $P = \{p_1, ..., p_n\} \in \mathbb{P}$, where each parameter p_i can have one or multiple possible values, $|p_i| \in \mathbb{N}$, $p_i \in \mathbb{R}$, and $\mathbb{P}$ is the parameter space, i.e., set of all possible P for A. Let $M = \{m_1, ..., m_n\}$ be a set of cost metrics, where each m_j is defined for A (e.g. planning time, path cost, etc).

For a particular planning algorithm A, the objective is to find a parameter combination $P^* = \{p_1^*, ..., p_n^*\}$, by using which the planning algorithm A will be able to perform a valid trajectory while optimising M,

$$P^* = \underset{P \in \mathbb{P}}{\arg\min}\, M(A, P). \tag{2}$$

A parameter tuning algorithm is a method that solves (2), preferably finding the optimal parameter set P^* in minimum time. However, a near-optimal solution can be returned if the tuning algorithm exceeds a predefined time limit, for instance, a two hours limit ($7,200$ s) is used in this work.

III. SOLUTION METHODS

The main difference between the four proposed methods is how they explore the parameter space $\mathbb{P}$. For example, random sampling explores it without following any rule nor based in any previous knowledge. The other three methods make use of existing information to explore the next configuration in the search space more intelligently, but according to different search strategies. All methods use a given planning algorithm A, optimise the set of metrics M, use the default configuration P_{def} as the initial best result known, and test a fixed number of parameter configurations $nConfs$.

Algorithm 1 describes the general tuning method common to all the solution methods, where the TERMINATE() function returns true if a termination criteria is met, e.g., exceeds time

Algorithm 1 General tuning method

```
Require: $A$, $M$, $P_{def}$
Ensure: $P^* \subset \mathbb{P}$
 1: $P^* = P_{def}$
 2: $nConfs = 0$
 3: # Search space exploration
 4: while NOT TERMINATE() do
 5:     $P' = $ NEWPARAMETERSET$(P^*, \mathbb{P})$
 6:     if $M(A, P') < M'(A, P^*)$ then
 7:         $P^* = P'$
 8:     end if
 9:     $nConfs = nConfs + 1$
10: end while
11: return $P^*$
```

limits. The NEWPARAMETERSET() is subject to different tuning algorithms (i.e. our four proposed methods).

Note that we chose the previous four methods incrementally in the following way: i) random sampling (rs) is a good starting point due to its simplicity; ii) bayesian optimisation (bo) is a solution widely used in robotics; iii) random forest (rf) has also shown to be an efficient method in the systems community; iv) and finally AUC-Bandit (ab) is commonly used as an optimisation heuristic in many domains. Next we provide more details about each solution method used.

A. Random Sampling

This method is based on selecting individual parameter configurations from the given search space randomly, where each configuration has the same probability of being chosen and is selected independently, i.e. any selected configuration might be repeated. The number of configurations selected, called $sample$, is a subset of the total search space, and each subset of k individuals also has the same probability of being chosen for the sample as any other subset of k individuals [14]. Note that the best solution within the sample typically improves as the sample size increases.

B. Bayesian Optimisation

This method is based on a sequential design of experiments strategy for global optimisation of black-box functions/systems (i.e. systems where an algebraic model is entirely unknown) based on using Bayesian statistics [15]. It is assumed that the black box can be queried either by simulation or using real data from experiments that provide a system output for specified values of system inputs.

Bayesian optimisation works well with functions that are very expensive to evaluate. This method uses a mathematical data-driven model (called surrogate model) that mimics the behaviour of another model as closely as possible while being computationally cheap(er) to evaluate. The model includes a utility function to guide the exploration. We use the Upper Confidence Bound (UCB) as our utility function, which has the free parameter κ that controls the balance between exploration and exploitation. We set $\kappa = 3$ which, in this case, makes the algorithm quite bold.

C. Random Forest

A decision tree represents a recursive binary partitioning of the input space, and uses a simple decision (a one-dimensional decision threshold) at each non-leaf node that aims at maximising an "information gain" function. A Random Forest predictor is a set of randomised decision trees.

Random Forest has been proved to be an efficient technique in the context of optimising parameters [16], [17]. In [17] the discovery of new points is done by using an active learning methodology. An initial set of random configurations are generated, then the predictor is used to generate the most promising configurations. Once they are run, those configurations are used to improve the quality of the predictor. This scenario is repeated as long as necessary. In [17] the size of the initial set was greater than 2000, and more than 100 new points were selected every iteration. Given the timing constraints which have been set in our experimental context, in our experiments we will limit the size of the initial set to one element and the number of new points to be selected to only one element as well.

D. AUC-Bandit

Multi-armed bandit with AUC (Area Under the Receiving Operator Curve) credit assignment is a comparison-based strategy implemented by OpenTuner [18]. OpenTuner is a generic tuning framework for building domain-specific multi-objective program auto-tuners. It features customisable configuration representations used to define the space of solutions. A key capability of OpenTuner is the use of ensembles of disparate search techniques simultaneously. Techniques which perform well will receive larger testing budgets and techniques which perform poorly will be disabled.

We used OpenTuner to build an auto-tuner for Motion Planning Algorithms. OpenTuner is composed of two main components, the `manipulator` function and the `run` function. The `manipulator` specifies the set of configuration parameters and their possible value ranges. The `run` executes the algorithm given those parameters and determines a score for them. Given those two components OpenTuner will then generate an auto-tuner for our given scenario.

IV. EVALUATION

In this section we define a set of robotic environments and then we analyse and compare the four different tuning methods using BKPIECE and RRT-Connect with the objective of answering the following research questions:

- *RQ1*: How well do the tuning methods perform in different environments, e.g. a realistic table-top-reaching scenario and randomly generated scenarios?
- *RQ2*: How robust are the selected tuning methods with respect to random scenarios of increasing complexity?
- *RQ3*: How does the parameter space of our problem look like when considering the solution cost?
- *RQ4*: How can the randomness in sampling-based algorithms affect our evaluation methodology and results?

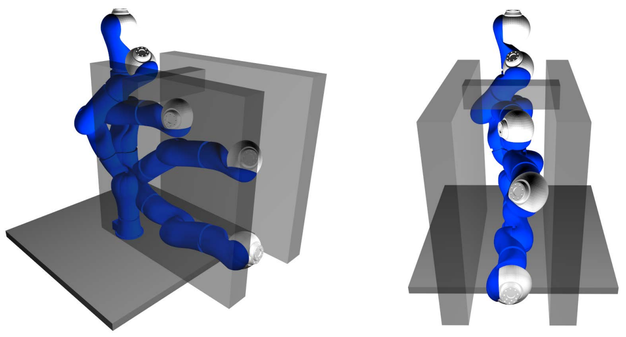

Fig. 1. Table-top-reaching scenario where a 7-DoF KUKA robot arm needs to move from upright posture to a goal configuration via a narrow passage. *left*: side view, *right*: front view.

TABLE I
PARAMETERS DEFINITION.

	Parameter	Range	Step	Default
RRT-Connect	Max steering distance (P1)	[0.1, 2.0]	0.1	1
	Simplification trails (P2)	[0, 10]	1	0
BKPIECE	Max steering distance (P1)	[0.1, 2.0]	0.1	1
	Border fraction (P2)	[0.05, 1.0]	0.05	0.5
	Failed exp. score factor (P3)	[0.05, 1.0]	0.05	0.5
	Min valid path fraction (P4)	[0.05, 1.0]	0.05	0.5
	Simplification trails (P5)	[0, 10]	1	0

A. Setup

In this section we describe the experiment set we used to evaluate the performance of the proposed tuning methods. Note that we focus on the planning time metric. A motion planning algorithm needs to plan a collision-free trajectory for a 7-DoF KUKA LWR robot arm to move from an initial to a goal configuration in two categories of environments. The first one is a typical table-top-reaching scenario that includes three objects, as shown in Fig. 1, and the second type is composed of random scenarios of increasing complexity, i.e. populated with 10, 50, 100, 200 and 300 ball obstacles at random locations, as shown in Fig. 2.

Two motion planning algorithms, BKPIECE and RRT-Connect, are selected in our evaluation. The reason for choosing these two algorithms is as follows. RRT-Connect is one of the most widely used sampling-based planners; thus finding the optimal parameters of RRT-Connect will bring great benefit to the robotics community. BKPIECE is also a commonly used algorithm; furthermore, in contrast to RRT-Connect that only has two parameters, BKPIECE can be configured by five different parameters which creates a huge parameter search space, making it a suitable algorithm for testing the tuning methods. The parameters of RRT-Connect and BKPIECE are highlighted in Table I. Due to the randomness present in sampling-based methods, we run $1,000$ trials for each problem and average the result.

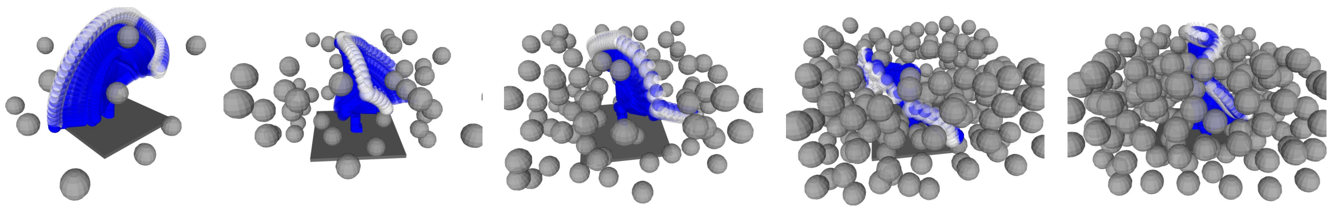

Fig. 2. Random scenarios: 10 objects (left), 50 objects (second left), 100 objects (center), 200 objects (second right), 300 objects (right).

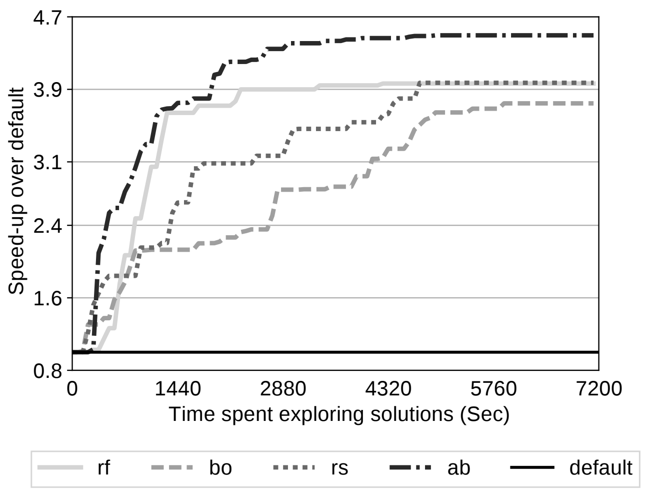

Fig. 3. Speedup of BKPIECE using different auto-tuning techniques over time for the table-top-reaching scenario.

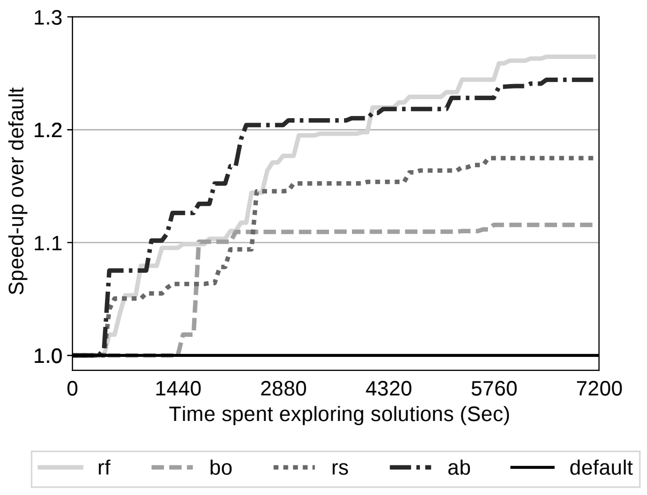

Fig. 4. Speedup of RRT-Connect using different auto-tuning techniques over time for the table-top-reaching scenario.

We have used the RRT-Connect and BKPIECE implementations from the EXOTica framework [19]. We developed the random sampling method and modified OpenTuner [18] to use the heuristics-based solution and also to integrate the random forest method in it. Finally, we adapted the bayesian optimisation implementation from [20]. The evaluations were carried out on Intel Core i7-6700K 4.0GHz CPU desktop machines with 32GB 2133MHz RAM.

B. Environment 1: Table-top-reaching scenario

Fig. 3 and Fig. 4 show the global speedup when using the optimal parameters found by the four methods discussed against the planning time of using the default parameter configuration ($speedup = 1$). Recall that we run each method for 7200 seconds (2 hours) and we repeated the experiment ten times. Therefore, each point in the graph represents the average value of ten runs. There are three important aspects to analyse in these graphs: i) the maximum average speedup achieved over the default configuration; ii) the differences among the proposed solution methods: iii) the computation time required to obtain solutions that improve the default performance significantly.

Answering *RQ1*, we can see in Fig. 3 that AUC-Bandit (ab) appears to be the best method at all times, achieving a maximum speedup of 4.5x over the default configuration. Random forest (rf) and random sampling (rs) provide a speedup of 4x after the 2 hours period, and bayesian optimisation (bo) roughly 3.8x. We also observe how after 1400 seconds (approx. 23 minutes), ab and rf are able to provide solutions that improve the default planning time by more than 3.5x, whereas bo and rs are only able to provide roughly 2x of speedup. When using RRT-Connect, as shown in Fig. 4, the speedup is much less than in BKPIECE. The optimal parameters found by random forest (rf) achieved a 1.26x speedup. However, considering that RRT-Connect only has 2 parameters, the improvement is still significant. Note that the methods that provide the best solution for both planning algorithms are different, and the user will use those.

The reason for the differences observed in Fig. 3 and Fig. 4 resides in the nature of each method, which defines how the solutions are explored along the time. For example, rs can select any configuration at any time, and some configurations in the search space are much slower to evaluate than others because they need more time to find a valid path. However, the other three methods explore the search space more intelligently, focusing more and more on good configurations as time progresses. This means that after the 2 hours period each method will be able to explore a different number of configurations, on average.

Tables II and III show the best parameter configuration, the planning time, and the total number of parameter sets explored after 2 hours (nConfs), on average, for each method. The tables also show the default configuration and its planning time. We see how ab and rf are the methods

TABLE II
BEST CONFIGURATIONS FOR BKPIECE AND THE ENVIRONMENT 1.

Method	Parameters (P1-P5)	Time(s)	nConfs
rs	{0.4, 0.1, 0.95, 0.7, 6}	0.31	15
ab	{0.3, 0.95, 0.2, 0.35, 3}	0.29	29
rf	{0.3, 0.65, 0.35, 0.9, 8}	0.34	17
bo	{0.3, 0.4, 0.7, 0.5 8}	0.34	17
default	{1, 0.5, 0.5, 0.5, 0}	1.45	-

TABLE III
BEST CONFIGURATIONS FOR RRT-CONNECT AND THE ENVIRONMENT 1.

Method	Parameters (P1-P2)	Time(s)	nConfs
rs	{1.6, 8}	3.58	17
ab	{1.5, 8}	3.53	19
rf	{1.5, 2}	3.32	24
bo	{1.4, 6}	3.65	9
default	{1, 0}	4.49	-

that explore the greatest number of configurations for both planners respectively, thus providing the best results. We also see how all the methods almost provide the same value for $P1$, which seems to be the parameter that most affects the planning time. Note that for BKPIECE, although the planning time of the best configuration for rs is lower than rf, on average they perform equally after two hours. Similarly, the best parameter set for rf and bo produces the same planning time but bo is worse than rf on average.

C. Environment 2: Random scenarios

To generate random scenarios of increasing complexity, we add a number of random objects in the environment. The greater the number of objects added, the more complex the scenario, and the more challenging it is to find a valid trajectory. Specifically, we consider five different scenarios each with 10, 50, 100, 200, and 300 objects randomly distributed, as shown in Fig. 2. For each problem, a random trajectory is first generated in free-space connecting the start and goal states, collision objects are then randomly populated into the environment without colliding with the trajectory. In this case, we guarantee that at least one valid solution exists for every problem. Since we run $1,000$ samples for each parameter configuration analysed, we generate $1,000$ random instances for each number of objects.

Tables IV and V show the difference for the planning time between the default configuration and the best solution found (by any of the methods) for the five scenarios considered and the two planning algorithms respectively.

Completing the answer for *RQ1*, there are two main observations in these tables: i) the difference for the planning time between the default and the best configuration increases with the complexity of the scenario; ii) the maximum difference (30% for 300 objects in BKPIECE, 22% for 300 objects in RRT-Connect) is much less than in a realistic scenario.

TABLE IV
PLANNING TIME OF BKPIECE USING THE DEFAULT AND THE BEST SOLUTION FOUND FOR EACH SCENARIO (IN SECONDS).

# objects	Parameters (P1-P5)	Method	Default	Best
10	{0.55, 1, 1, 0.05, 5}	bo	0.06	0.05
50	{0.6, 0.4, 0.5, 0.1, 4}	bo	0.12	0.10
100	{0.7, 0.35, 0.5, 0.1, 0}	ab	0.24	0.18
200	{0.7, 0.8, 0.95, 0.3, 5}	ab	0.87	0.67
300	{0.6, 0.9, 0.9, 0.7, 8}	rf	2.06	1.43

TABLE V
PLANNING TIME OF RRT-CONNECT USING THE DEFAULT AND THE BEST SOLUTION FOUND FOR EACH SCENARIO (IN SECONDS).

# objects	Parameters (P1-P2)	Method	Default	Best
10	{1.5, 4.0}	ab	0.004	0.003
50	{0.8, 6.0}	rf	0.015	0.012
100	{1.5, 3.0}	rs	0.063	0.071
200	{0.8, 4.0}	rf	0.995	0.553
300	{1.7, 7.0}	rf	2.177	1.703

These results make sense as we are testing each parameter configuration across $1,000$ different random instances, thus we are actually solving the average case (i.e. training the scenario) for each number of objects.

Tables IV and V also show the parameter configuration that provides the best planning time for each scenario along with the method that obtained it. For BKPIECE, we see that for scenarios of low (10 objects) and medium complexity (50 and 100 objects) bayesian optimisation (bo) and AUC-Bandit (ab) provide the best results. However, for scenarios of high complexity (200 and 300 objects) AUC-Bandit (ab) and random forest (rf) are the best methods. In addition, as it happened with the table-top-reaching scenario, we observe that the value of $P1$ for BKPIECE falls within a small interval. However, for RRT-Connect we observe greater differences for $P1$ and random forest (rf) is the best method for three of the scenarios. Therefore, we see how the improvement when we solve a more realistic environment (e.g. the table-top-reaching scenario) is significantly higher than for the general case, but even if we randomly generate the number of objects present in the scenario we obtain a solution that performs better than the default.

D. Robustness analysis: Training vs Testing

We now analyse the robustness of the methodology proposed by running the best parameter configuration for each type of random scenario (obtained in the training phase) in all the random scenarios including itself (we call it *matching scenario*). These *testing* experiments need to be repeated only once, since the training phase requires to run each experiment ten times. Small differences among different runs of the same

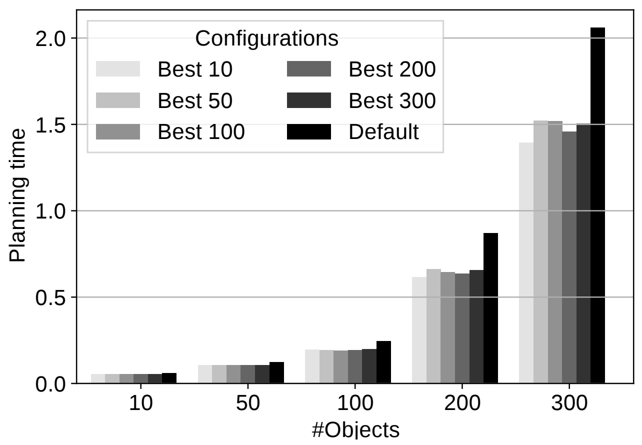

Fig. 5. Robustness analysis: for each random scenario (10, 50, 100, 200 and 300 objects), we run BKPIECE with its default configuration and with the best configuration obtained for each scenario (including itself).

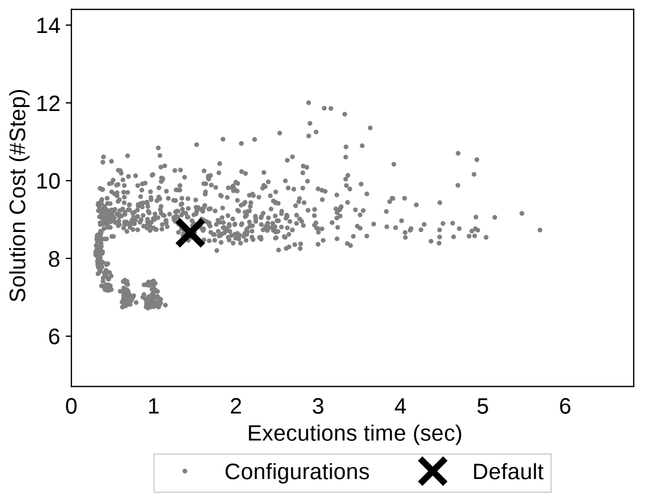

Fig. 6. All the configurations of BKPIECE explored so far.

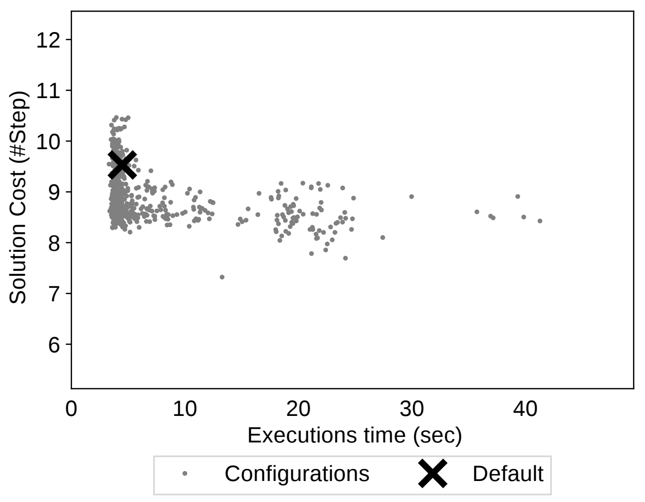

Fig. 7. All the configurations of RRT-Connect explored so far.

testing experiment can only be attributed to the inherent *randomness* present in sampling-based algorithms.

Answering *RQ2*, Fig. 5 shows the results of these experiments for BKPIECE. Comparing the values obtained for the *matching scenarios* with the values in Tables IV and V we see that the results are almost the same, which demonstrates the robustness of the best solution found for each type of random scenario. Fig. 5 also shows that for scenarios of low and medium complexity (i.e. 10, 50, and 100 objects) the best configuration for each type of scenario provides the lowest planning time, although the best configurations for other scenarios can provide similar results. For complex scenarios (200 and 300 objects) the differences among the best configurations are more apparent. However, we see how the best configuration for 10 objects provides the best result also for the complex scenarios. This is an excellent result, since it means that we can train scenarios on small number of objects (with lower training times) to give good performance on more complex scenarios with more obstacles, and also outperforming the default configuration.

E. Parameter space of the algorithms

In order to answer *RQ3*, we now analyse both the planning time and the path cost (given by its length) of each solution. Fig. 6 and Fig. 7 show the data points obtained for all the experiments performed for the table-top-reaching scenario according to the two metrics. The ”X” represents the location of the default configuration in the parameter space.

As we see in Fig. 6, the data points for BKPIECE are quite scattered, ranging from 0.3 to 6 seconds for the planning time and from 7 to 12 for the path cost. We also observe that there are many data points that improve the default configuration. More specifically, the points that provide the best planning time also improve the solution cost by 10-12%. For RRT-Connect, as shown in Fig. 7, the data points are more dense. While there was no significant reduction in planning time, the solution cost can be optimised up to 15%.

F. Randomness analysis

Evaluating sampling-based algorithms in robotics typically implies to repeat every experiment a great number of times in order to obtain an average value (and variance) for some metric that can give the user confidence in the system behaviour. Common repetition values for sampled-based experiments are $1,000$ or even $10,000$ times. However, this number of repetitions is typically performed systematically, without analysing if it is actually required for the corresponding system and experiment. As we pointed out previously, each parameter configuration is tested $1,000$ times. We now study if this number was actually required. Note that increasing/decreasing this repetition value generates higher/lower evaluation times for each configuration tested.

To answer this question, we compared for any number of executions of a specific configuration, the deviation of the average execution time against the average execution time for $1,000$ executions of the same configuration. Fig. 8 shows this

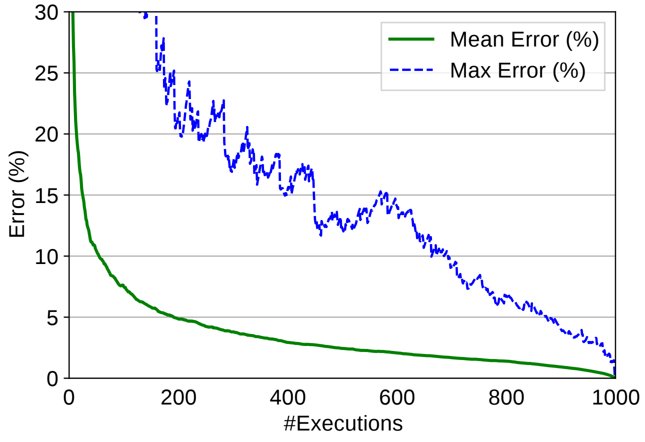

Fig. 8. Randomness of BKPIECE for the table-top-reaching scenario.

deviation considering all the experiments performed for the table-top-reaching scenario and BKPIECE. The graph shows two curves, which represent the maximum (Max Error) an average deviation (Mean Error) of the current average value (obtained considering the number of repetitions so far).

As we see in the graph and answering *RQ4*, for the planning time after 300 repetitions the average deviation is under 5%, going to 2.5% for 600 repetitions, compared with $1,000$ repetitions. Therefore, depending on the maximum deviation that (the user of) a given application can tolerate, we could reduce the number of repetitions and thus the evaluation time of each experiment dramatically. In the table-top-reaching scenario, considering the planning time and assuming an average error tolerance of 5% we would reduce the evaluation time by more than 2/3.

V. CONCLUSIONS

We have analysed and compared four tuning methods (random sampling, bayesian optimisation, random forest and AUC-Bandit) for configuring the parameters of two sampling-based motion planning algorithms, RRT-Connect and BKPIECE, with the objective of minimising the overall planning time. We have proposed a real-world table-top-reaching scenario and we have seen how all the proposed methods provide a better configuration than the default one, where the best method (AUC-Bandit) clearly outperforms the rest of solutions, reducing the planning time provided by the default configuration by up to 4.5x. We have also analysed the robustness of our methodology across a set of randomly generated scenarios, and we have found that we can train random scenarios with a small number of objects in a short time and obtain excellent performance in scenarios with many objects that require long training times. Finally, to further reduced the required evaluation times we have seen that the number of samples required for the experiments can be dramatically reduced within low error rates. Therefore, the proposed tuning methodologies may be interesting for users, as they provide significant performance improvements for specific scenarios with reasonable computation times.

ACKNOWLEDGMENTS

This work was supported by the AnyScale Applications project under the EPSRC grant EP/L000725/1, and partially by the EPSRC grant PAMELA EP/K008730/1.

REFERENCES

[1] S. M. LaValle, *Planning Algorithms*. New York, NY, USA: Cambridge University Press, 2006.
[2] S. M. Lavalle, “Rapidly-exploring random trees: A new tool for path planning,” Tech. Rep., 1998.
[3] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.
[4] M. Elbanhawi and M. Simic, “Sampling-based robot motion planning: A review,” *IEEE Access*, vol. 2, pp. 56–77, 2014.
[5] J. J. Kuffner and S. M. LaValle, “Rrt-connect: An efficient approach to single-query path planning,” in *IEEE International Conference on Robotics and Automation.*, 2000, pp. 995–1001.
[6] I. A. Sucan and L. E. Kavraki, “Kinodynamic motion planning by interior-exterior cell exploration,” in *Algorithmic Foundation of Robotics VIII, Selected Contributions of the Eight International Workshop on the Algorithmic Foundations of Robotics, WAFR 2008, Guanajuato, México, December 7-9*, 2008.
[7] D. D. Dunlap, C. V. Caldwell, E. G. C. Jr., and O. Chuy, *Motion Planning for Mobile Robots Via Sampling-Based Model Predictive Optimization*. InTech, December 2011.
[8] J. Luo and K. Hauser, “An empirical study of optimal motion planning,” in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept 2014, pp. 1761–1768.
[9] R. Burger, M. Bharatheesha, M. van Eert, and R. Babuka, “Automated tuning and configuration of path planning algorithms,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 4371–4376.
[10] M. Moll, I. A. Sucan, and L. E. Kavraki, “Benchmarking motion planning algorithms: An extensible infrastructure for analysis and visualization,” *IEEE Robotics Automation Magazine*, vol. 22, no. 3, pp. 96–102, Sept 2015.
[11] J. Cano, D. R. White, A. Bordallo, C. McCreesh, A. L. Michala, J. Singer, and V. Nagarajan, “Solving the task variant allocation problem in distributed robotics,” *Autonomous Robots*, Apr 2018.
[12] J. Cano, D. R. White, A. Bordallo, C. McCreesh, P. Prosser, J. Singer, and V. Nagarajan, “Task variant allocation in distributed robotics,” in *Proceedings of Robotics: Science and Systems (RSS)*, June 2016.
[13] J. Cano, A. Bordallo, V. Nagarajan, S. Ramamoorthy, and S. Vijayakumar, “Automatic configuration of ROS applications for near-optimal performance,” in *2016 IEEE/RSJ International Conference on IntIntelligent Robots and Systems (IROS)*, October 2016.
[14] J. Tabor, D. Yates, and D. S. Moore, *The Practice of Statistics*. W. H. Freeman; 5 edition, 2014.
[15] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, “Taking the human out of the loop: A review of bayesian optimization,” *Proceedings of the IEEE*, vol. 104, pp. 148–175, Jan 2016.
[16] F. Hutter, H. H. Hoos, and K. Leyton-Brown, “Sequential model-based optimization for general algorithm configuration,” in *Learning and Intelligent Optimization*, C. A. C. Coello, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 507–523.
[17] B. Bodin, L. Nardi, M. Z. Zia, H. Wagstaff, G. S. Shenoy, M. Emani, J. Mawer, C. Kotselidis, A. Nisbet, M. Lujan, B. Franke, P. H. J. Kelly, and M. O’Boyle, “Integrating algorithmic parameters into benchmarking and design space exploration in 3d scene understanding,” in *2016 International Conference on Parallel Architecture and Compilation Techniques (PACT)*, Sept 2016, pp. 57–69.
[18] J. Ansel, S. Kamil, K. Veeramachaneni, J. Ragan-Kelley, J. Bosboom, U.-M. O’Reilly, and S. Amarasinghe, “Opentuner: An extensible framework for program autotuning,” in *Int. Conference on Parallel Architectures and Compilation Techniques (PACT)*, August 2014.
[19] V. Ivan, Y. Yang, W. Merkt, M. Camilleri, and S. Vijayakumar, “Exotica: An extensible optimization toolset for prototyping and benchmarking motion planning and control,” in *Robot Operating System (ROS): The Complete Reference*, 2019, vol. 3, pp. 211–240.
[20] “Bayesian optimization,” https://github.com/fmfn/BayesianOptimization, accessed: 2017-09-12.