



Deposited via The University of York.

White Rose Research Online URL for this paper:

<https://eprints.whiterose.ac.uk/id/eprint/242492/>

Version: Accepted Version

---

**Proceedings Paper:**

ULRICH-OLTEAN, FELIX, BATE, IAIN JOHN, LIU, PENGCHENG et al. (2026) PAMPR: Predictability-Aware Motion Planning for Robots. In: Annual Conference Towards Autonomous Robotic Systems:proceedings. 27th Annual Conference Towards Autonomous Robotic Systems, 07-09 Sep 2026 Lecture Notes in Computer Science. Springer, GBR.

---

**Reuse**

This article is distributed under the terms of the Creative Commons Attribution (CC BY) licence. This licence allows you to distribute, remix, tweak, and build upon the work, even commercially, as long as you credit the authors for the original work. More information and the full terms of the licence here:

<https://creativecommons.org/licenses/>

**Takedown**

If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing [eprints@whiterose.ac.uk](mailto:eprints@whiterose.ac.uk) including the URL of the record and the reason for the withdrawal request.

# PAMPR: Predictability-Aware Motion Planning for Robots

Felix Ulrich-Oltean<sup>1</sup>, Iain Bate<sup>1</sup>, Pengcheng Liu<sup>1</sup>, Shibao Yang<sup>1</sup>

**Abstract**—Motion planning algorithms are crucial to the effective operation of robots; however, many motion planning algorithms exist, and they have complementary strengths when applied to different planning scenarios. The goal of this work is to increase predictability in the context of motion planning across a set of targets. A challenge is that each task and each planner have different levels of variability in the time they need for each target. The sub-goals of this work are therefore to shorten the time to complete the sequence at the same time as reducing the variability of the completion time. This combination helps reduce the likelihood that a deadline is missed. Our framework for predictability-aware motion planning for robots (PAMPR) uses machine learning to predict the planning and execution time for each available algorithm and query; these predictions are then used by a constraint solver to find an optimal sequence of targets and algorithm choices. We use an assistive robot arm example to show that this approach can lead to faster and more predictable completion times for the collection of tasks. Compared to the single best planner according to the training data, PAMPR reduces the right tail of the distribution by 47% and is twice as fast on average.

## I. INTRODUCTION

Motion planning is a crucial aspect of robotics, ensuring a safe path for a robot between two given positions or poses and providing the low-level instructions needed to achieve the transition. A wide variety of motion planning algorithms (or simply *planners*) have been developed, with many advances made over recent decades. Today’s robotics community has access to a large collection of state-of-the-art planners, many of which are consolidated in the Open Motion Planning Library (OMPL) [1], providing a common basis for benchmarking and comparative evaluation.

Different motion planners exhibit complementary strengths. Some planners achieve fast solutions on average but suffer from high variability, while others provide more consistent but slower performance. This trade-off is critical when reliability is required. In particular, planners with heavy-tailed execution time distributions may perform poorly in worst-case scenarios despite good average performance.

Typically, robots are used to carry out a sequence of motions in order to achieve a task. Planning algorithms have complementary strengths in terms of how resource-intensive they are to run and how likely they are to return a viable plan for a given query in a reasonable time frame. Most planners have a stochastic element and hence the same planner can return a variety of different outcomes for the same query. For many users and applications, predictability is a major consideration if a system is to be trusted and deemed reliable.

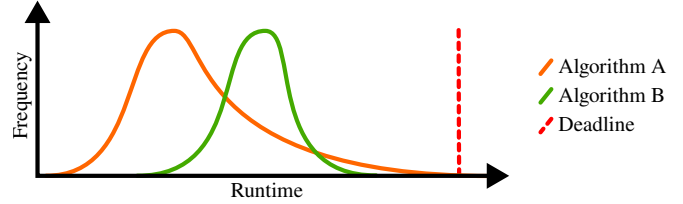


Fig. 1. Performance (distribution of execution times) of two hypothetical algorithms over the same inputs. Algorithm A is faster on average, but has a longer tail; algorithm B is slower on average but has a lower worst-case performance and less variability.

Consider a hypothetical task for which two algorithms are available. When the two algorithms are run several times on the same task, the completion times might form the distributions shown in Figure 1. If we were chiefly (or only) concerned with average running times, we would choose algorithm A. However, if a deadline was involved, we would prefer algorithm B as its runtimes are more symmetrical, with a more predictable range of times. B also has a much higher likelihood of its worst case finishing before the deadline. An everyday analogy would be choosing between a multi-lane motorway which is not always the quickest, but more predictable versus a narrower road which is often faster but prone to occasional accidents which close the whole road. In this paper, deadlines are treated as *hard* deadlines where there is no point completing the task(s) after the deadline. Continuing the analogy, there is no point arriving at an interview after the start time so minimising that risk is more important than the route that gets us there the earliest.

However, if the overall task involved two separate sub-tasks which could both be addressed by algorithms A and B, then it may be that for the second sub-task algorithm B has a lower average but is less predictable. Therefore the overall plan may be better off using a different algorithm for each task. When making this decision though, it should be considered that the overheads of switching algorithm may not be negligible.

In this work we show that when we select from a portfolio of relevant algorithms, we are able to improve both the average running time and the predictability or robustness of the generated plans for multi-step tasks. We go on to show that compared to the single best planner chosen from the training data, our solution, called PAMPR, is more than twice as fast on average and reduces the right tail of the distribution by 47%. We employ machine learning (ML) techniques as well as constraint programming (CP) to construct our solution.

<sup>1</sup>Department of Computer Science, University of York, York, UK

## Contributions and Structure

In this paper we present a solution built on well-studied machine learning and algorithm selection techniques. Our specific contributions are:

- 1) a hybrid (symbolic and statistical) AI method for joint optimisation of task ordering and algorithm selection under uncertainty;
- 2) a novel predictability-aware target labelling scheme for ML regression;
- 3) useful features for the ML regression;
- 4) an empirical study demonstrating the gains in time and predictability of our approach versus a reasonable default;
- 5) the software with which to reconstruct or modify our simulated test environment and carry out the experiments in this paper<sup>1</sup>.

The rest of the paper is structured as follows. Section II describes **related work** and positions our problem and proposed solution. We briefly define the specific **problem** we are addressing in Section III. We go on to describe our approach and details of our **method** in Section IV, evaluating with an **empirical study** in Section V. Finally, we summarise and suggest next steps in Section VI.

## II. RELATED WORK

### A. Complementarity

No single planner performs best across all scenarios, but instead the user needs to choose planners according to their suitability for particular tasks. In [2] the performance of several planning algorithms is benchmarked with several robot arms across a variety of scenarios with different amounts of clutter – the algorithms are deployed with a number of different hyperparameter settings. The authors investigate *robustness*, defining it in terms of the sensitivity of a planner’s performance to the setting of its *range* parameter.

A recent comprehensive survey of sampling-based planners [3] investigates the interplay between hyperparameters, planning time and path length in different task and dimensionality settings. Once again the conclusion is that different planners are better suited to different challenges.

In this work we show that the ability to choose different planning algorithms is useful even when performing similar tasks within the same environment (see Section III). However we recognise that changing the planner being used to achieve a faster and more predictable overall outcome doesn’t come for free. On a resource-constrained platform, it is quite realistic to envisage various overheads, e.g. for safety having to stop moving while the software is reconfigured, for reasons of limited memory the existing software may need to be offloaded before the new software can be loaded and activated. A classic example from real-time systems is Sha et al.’s work on mode change protocols [4] where a significant cost is incurred when changing modes. Our work allows this switching overhead to be considered as part of generating the overall plan.

### B. Algorithm Selection and Configuration

Choosing the right algorithmic tool for the job is a well studied problem. Known as the Algorithm Selection Problem, this perspective was formalised by Rice [5] as follows:

given a set of problems  $x \in \mathcal{P}$ , a set of available algorithms  $A \in \mathcal{A}$  and a performance measure  $p(A, x)$ , find a selection mapping  $S : x \mapsto A$  which maximises the performance.

The performance measure is very often runtime, especially when many instances of the same problem class are to be solved; an improvement in average runtime over the default algorithm choice can bring significant financial or practical benefits. We are interested in predictability, which may be defined by having a low variance in the runtime or a bounded tail.

The algorithm configuration problem is closely related, essentially being a maximisation problem across the hyperparameters of an algorithm for the problem instance at hand. Where alternative algorithms exist for the same problem class, and each algorithm has several configuration hyperparameters, the algorithm selection problem can be subsumed into a large algorithm selection problem, with each (algorithm, configuration) pair effectively being treated as a separate algorithm. This is how we proceed in this work.

Kerschke et al. provide a helpful review [6] of automated algorithm selection and configuration across many problem types. They also describe the commonly used methods for comparing the performance of automatically selected algorithms versus various baselines. Common ideas include:

- the *Single Best* (SB) is the one algorithm which is known to be the best performing overall, generally because it performed best on the training set of problems
- the *Virtual Best* (VB) also sometimes termed the *Oracle* is imagining we had full foresight, i.e. always choosing the best performing algorithm for any problem in the test/evaluation set
- when an algorithm does not finish within a set time limit, a *timeout penalty* is applied; this is most often a multiple of the time limit, so *PAR10* would mean that when an algorithm times out, it is awarded 10 times the time limit for the purpose of comparing to other algorithms or algorithm selectors

### C. Multiple Targets

There are scenarios where a robot is required to carry out a set of tasks within the same environment. We show that by delegating the sub-tasks to different planners, the quality of the overall plan can be improved, in terms of average time to plan and execute, as well as the probability of successful completion.

In integrated task and motion planning (TAMP) [7], [8] the aim is to plan a sequence of actions to achieve a goal. In our work we are more narrowly focused on combining a set of queries for which the order doesn’t necessarily matter, but may become important when the overhead of switching planners is taken into consideration.

<sup>1</sup><https://github.com/felixvuo/pampr-software>

An interesting problem somewhat analogous to ours is tackled by Sripathy et al. [9]; the context is autonomous vehicles and they describe a “ladder” of available models of human behaviour, with higher models requiring higher computation cost but promising to yield a better reward in the quality of prediction they give. Their work also mentions the cost of switching models and they introduce a hyperparameter  $\lambda$  which determines how conservative or aggressive the system is in relation to switching models.

In our setting, the algorithm selection problem is extended to a sequence of interdependent robot motion planning tasks. We must therefore determine not only which motion planning algorithm to use for each task, but also the order in which tasks should be executed. This leads to a combinatorial optimisation problem over both task sequences and algorithm choices. The ordering of these targets affects both planning and execution costs. When combined with planner selection, this results in a multi-modal optimisation problem akin to a travelling salesperson problem with multiple cost models.

#### D. Constraint Programming

As discussed previously, we need to decide the order in which to reach the given poses, as well as which planner to employ for each step. Constraint programming (CP) offers the ability to describe a combinatorial problem in a natural high-level way, and then have access to a variety of solving technologies to search for an actual solution. For example, the Savile Row constraint modelling assistant [10] takes constraint models written in the *Essence Prime* language and encodes them (with sophisticated reformulations) for back end solvers of different types: clause-learning constraint solvers like Chuffed, Boolean Satisfiability solvers like Kissat, as well as SMT, MIP and MaxSAT. The MiniZinc [11] ecosystem similarly provides a high-level language with greater flexibility, as well as a range of output formats for eventual solving. Google’s OR-Tools suite [12] has consistently won constraint solving competitions over recent years. It uses portfolio approaches to start a number of solvers on multi-core systems as well as many sophisticated reformulations and optimisations.

We decide to use a CP approach as it easily accommodates the combinatorial nature of our challenge, which requires choosing an itinerary between poses, as well as which planner to use, bearing in mind the penalty of switching planners between subsequent steps. We also gain from the ability to combine different solving strategies, which OR-Tools in fact provides out of the box.

Furthermore, CP would easily allow the addition of other constraints in the future, such as inter-task dependencies.

### III. PROBLEM STATEMENT

The **context** of the problem we are addressing here is that of a robot arm in a cluttered but static environment.

The **task** is to reach each of a predefined set of poses, starting and finishing in a “home” pose. This requires the proposed system to choose an order of poses and to select an algorithm to plan the motion between subsequent poses.

The aim is to complete the task as quickly and predictably as possible; in other words, the proposed system **must**:

- ensure the robot completes the task without colliding with itself or any obstacles
- complete the overall task within a given deadline

Furthermore, it **should**:

- avoid frequently switching planning algorithms between steps
- complete the overall task quickly on average
- minimise the variation (especially the right tail) in completion times for the same task over several attempts

To demonstrate the need for such a system, we present in Figure 2 two queries from our experiments (which are described in detail and analysed in Section V). We selected two of the hardest queries for illustration, i.e. queries which took longest on average to plan across the algorithms in our portfolio. We see that for the first query (from pose 20 to pose 3) nine algorithms are able to produce some plans in 20 attempts within the allotted time of 60 seconds. However, only two algorithms are producing plans for every attempt, and doing it faster than the others. The second query is even more challenging, with no algorithm reliably producing a plan every time. In this second case, the three algorithms which perform best (*lazyprm\*0*, *lazyprm\*3*, *lazyprm\*1*) were not the top performers in the first query. The plots also show that the algorithms have different variations in planning time and resulting execution time.

### IV. COMBINING MACHINE LEARNING AND CONSTRAINT PROGRAMMING FOR PAMPR

#### A. Outline

The structure of our proposed solution is illustrated in Figure 3. In brief,

- features are extracted from the simulated world;
- ML regressors are trained to predict the time and variability for each planner for every possible query between the poses in the training set;
- for the test set, the ML model predicts the performance of each planner for each possible query in the test set;
- a constraint solver chooses a sequence of poses and an algorithm for each step between successive poses.

In this section we describe the steps above at a high level, giving further implementation details in Section V.

#### B. Machine Learning for Performance Prediction

*Training Data:* The system is trained on a set of queries, using all the relevant algorithms. We automatically generate a set of poses in the vicinity of a complex object, ensuring that the poses are each valid (i.e. do not collide with the object and avoid robot self-collision).

Due to the variability in the running time of the algorithms and in the length of the resulting plans, each query should be run several times to obtain a distribution of times for each (query, algorithm) pair. We construct a dataset by running each planning algorithm with different seeds on each possible pairwise query between the generated poses,

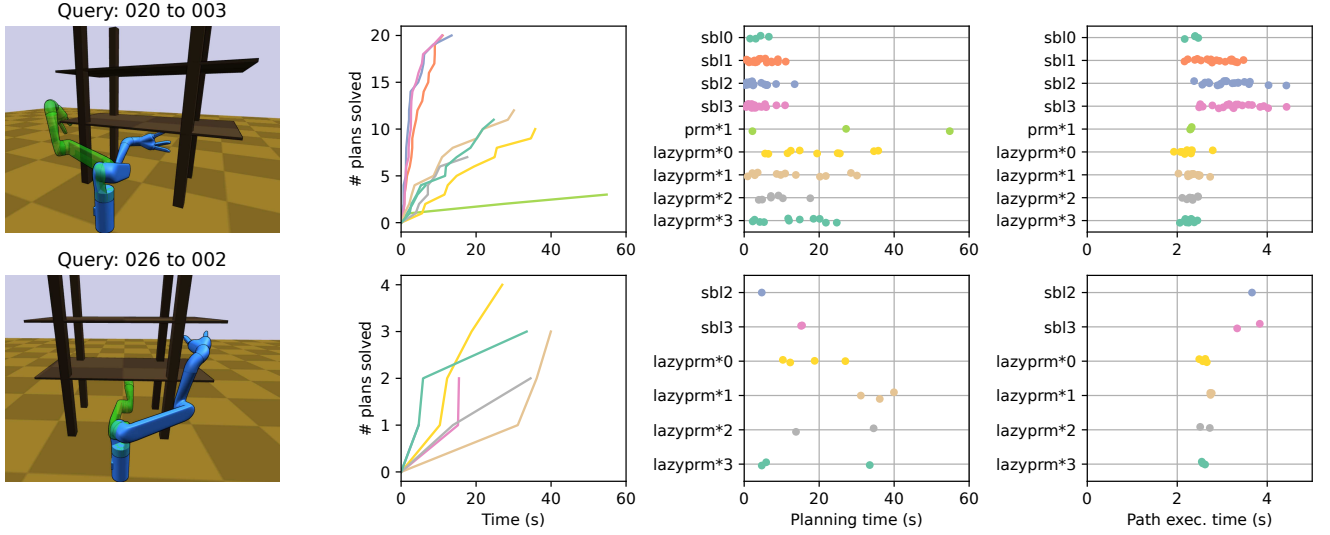


Fig. 2. Comparison of planner performance on two of the harder queries in our dataset, where each query is executed 20 times. The first panel in each row shows the start pose in blue and the goal pose in green; the second panel plots the number of times a plan has been found by any given time up to the timeout of 60 seconds; the third panel shows the planning time; the final panel shows the resulting path execution times for the plans created.

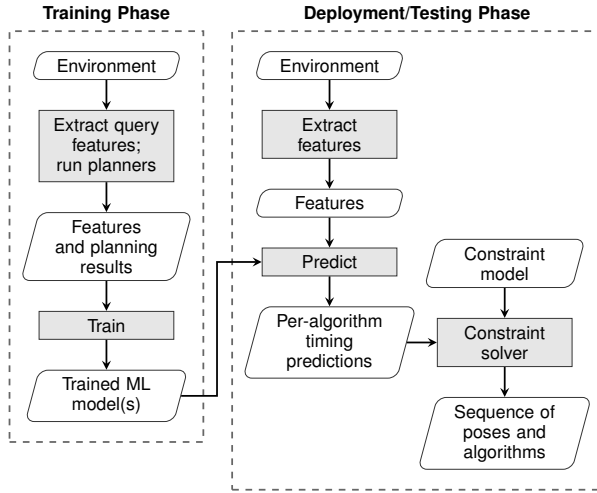


Fig. 3. Overview of our system which uses ML regressors to estimate the performance of each available planner on the set of possible tasks; a constraint solver uses these estimates as parameters to a constraint model to decide on a sequence of tasks and algorithm choices.

recording the planning time and the simulated execution time of the generated plan, if indeed a plan has been returned within the allocated time.

Whenever a plan is not returned within the given deadline, we record a penalised time, which is  $k \times \text{timeout}$  where  $k$  is a non-negative real number.

**Features:** We extract the following features for each pairwise query between the poses:

- the co-ordinates of the poses in the robot’s configuration space
- the co-ordinates in 3D space of the robot’s end effector
- the minimum distance in 3D space between each of the robot links and the start and goal pose

Additionally, we record:

- the planner type (e.g. RRT\*), one-hot encoded
- algorithm parameters such as: perturbation radius, KNN neighbourhood size, whether the search is bidirectional, the restart frequency

**Target Labels:** The ML component of our solution is intended to support making a good choice of algorithm given a particular planning scenario. On a simplistic level, we want to choose the best planning algorithm configuration for each step. However, this choice cannot be made in isolation, because of the switching penalty. Therefore, the ML component in PAMPR predicts the amount of time each algorithm would take on any given query, letting the downstream CP component decide which order to do the queries in, taking into account the switching penalty and any other constraints.

Another element to consider is the predictability of the planners’ performance. As illustrated in Figure 2, the combined planning and execution time of any planner can vary widely; an algorithm may even not return a result in a reasonable time. We address this aspect of the challenge in 2 ways:

- 1) As well as training a regressor on the combined planning and execution time, we also train a separate regressor to predict the time + a “tail”. In our experiments, we attempt each query several times with each planner, giving us a distribution of results. We define the tail as the 50-95th inter-percentile difference – this value is then added to each individual time. This (time + tail) measure effectively applies a penalty to the algorithms with more variation, specifically with long tail performance, i.e. prone to produce occasional results much longer than the average.
- 2) We use a standard practice from algorithm selection,

which is to apply a timeout penalty when an algorithm has not returned a plan in the allotted time.

Note that as well as the *time* and (*time* + *tail*) label we also implemented a (*time* + *standard deviation*) label; however, the distribution of times is not always Gaussian, and the standard deviation is sensitive to both fast and slow extremes, whereas we principally want to avoid the long tail. Therefore we do not include this third label in the work presented.

*ML Model:* In terms of the machine learning model, we use random forest regression [13]; random forest is a widely applicable ML algorithm, which can be applied to different data types and avoids overfitting by using an ensemble of decision trees, each of which is trained on a subset of the training data. In early experiments we also tried other ML algorithms, such as gradient boosted trees and extremely random trees but random forest gave the best compromise between complexity, training time and accuracy.

### C. CP for Deciding Tour and Planners

#### Definition of Constraint Satisfaction and Optimisation:

For the purpose of this paper, we define a constraint satisfaction problem (CSP)  $\mathcal{P}$  as a tuple containing three sets  $\mathcal{P} = \langle X, D, C \rangle$ , where:  $X$  is a set of decision variables  $x_1 \dots x_n$ ;  $D$  is a set of domains, with  $D(x_i) = \text{Dom}(X_i)$ ;  $C$  is a set of constraints each of which imposes restrictions on the allowed values assigned to a subset of  $X$ . A constraint optimisation problem (COP) additionally introduces a function of a subset of decision variables which is to be minimised or maximised. A solution to a CSP is a complete assignment of values to all variables  $x_1 \dots x_n$  such that every variable  $x_i$  is assigned a value from its domain  $D(x_i)$  and all constraints are satisfied (they evaluate to true). If multiple satisfying assignments exist, then the problem has multiple solutions. When no satisfying assignment exists, the problem is said to be unsatisfiable. For a COP, the solutions are the sets of assignments which satisfy the CSP and additionally minimise (or maximise) the result of the objective function.

*Constraint Model:* The problem of ordering the queries and algorithms is essentially a multi-modal version of the travelling salesperson problem, with an optional penalty for switching modes (algorithms).

The constraint optimisation problem can be described as follows.

Given:

- $P = \{p_1, \dots, p_n\}$ : The set of  $n$  poses.
- $p_h \in P$ : The “home” pose for the robot arm.
- $A = \{a_1, \dots, a_m\}$ : The set of  $m$  available algorithms.
- $T(a, i, j) \in \mathbb{Z}^+$ : The planning and execution time of algorithm  $a \in A$  for the query  $p_i \rightarrow p_j$ .
- $c \in \mathbb{Z}^+$ : The cost of switching algorithms.

Decide:

- $\text{tour} : \{0 \dots n\} \mapsto P$ : The sequence of places to visit.
- $\text{algo} : \{1 \dots n\} \mapsto A$ : The sequence of algorithms.
- $\text{switch} : \{1 \dots n\} \mapsto \{0, 1\}$ : Whether the algorithm has been switched at each step.
- $\text{time} : \{1 \dots n\} \mapsto \{0 \dots \max(T)\}$ : The time taken for each step.

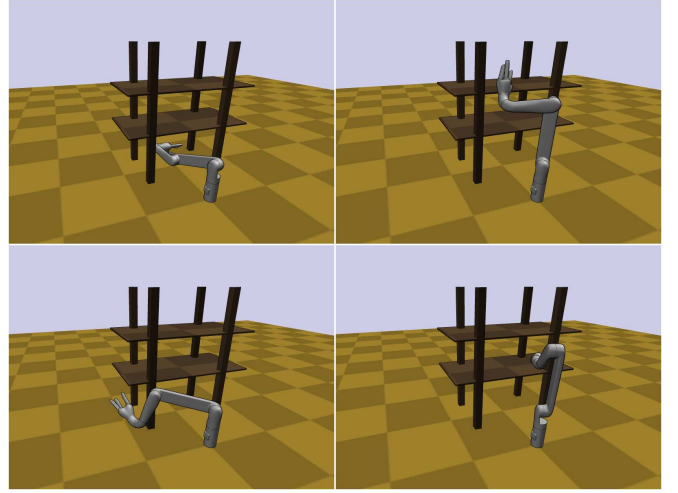


Fig. 4. A sample of the 40 poses generated

Subject to:

- $\text{tour}(0) = \text{tour}(n) = p_h$ : The tour starts and ends on the “home” pose.
- $\text{AllDifferent}(\{\text{tour}(1) \dots \text{tour}(n-1)\})$ : Each pose in-between is unique.
- $\text{switch}(i) = 1 \leftrightarrow \text{algo}(i-1) \neq \text{algo}(i), i \in 2 \dots n$ : Record switching of algorithm.
- $\text{switch}(1) = 1$ : Enforce cost of loading initial algorithm
- $\text{time}(i) = T(\text{algo}(i), \text{tour}(i-1), \text{tour}(i)), i \in 1 \dots n$ : Assign the time taken by the chosen algorithm for each step.

Minimising:

$$\sum \text{time} + c \sum \text{switch}$$

We model the problem using integer domains and solve using a finite integer domain constraint solver. In order to represent the planning and execution times, which are reals, we use a *granularity* parameter to split a real second into discrete time steps for the CP model.

## V. EMPIRICAL STUDY

### A. Data Set

We use the *Klamp't* Software Suite [14], [15] which supports modelling, planning and simulation for robots. We generate 40 poses automatically, ensuring that the robot is not colliding with itself or the scene in each case, and that the poses are not too close to each other in configuration space. A sample of poses is shown in fig. 4. The robot being simulated is the *Jaco Assistive Arm* [16], which has 7 links and a 3 finger attachment giving 10 degrees of freedom and has a reach of approximately 90 cm.

We use 5 planning algorithms implemented natively by *Klamp't*. These are:

- **SBL**: The Single-Query Bidirectional Lazy Planner [17], which generates two trees from the start and goal and defers collision checking until a candidate path is found, yielding rapid first solutions in cluttered environments.

- **RRT\***: Rapidly-exploring Random Trees (\* optimising) [18], which rewires the tree to minimise cumulative path cost, guaranteeing asymptotic optimality and suiting single-query problems where path quality is important.
- **PRM\***: Probabilistic Road Maps (\* optimising) [18], which applies analogous rewiring to road map construction and performs strongly in multi-query scenarios, albeit with a greater preprocessing cost.
- **Lazy-RRT\***: RRT\* with Lazy Collision Checking [19], which defers edge validation whilst preserving asymptotic optimality, reducing expensive collision checks and improving planning times in cluttered scenes.
- **Lazy-PRM\***: PRM\* with Lazy Collision Checking [19], which applies the same lazy-evaluation principle to PRM\*, lowering validation overhead whilst retaining optimality guarantees.

This portfolio of algorithm consists of all the natively implemented optimising planning algorithms in *Klamp't* which were competitive in our simulated scenario. For example, *Klamp't* also implements FMM\* (an optimising fast marching method algorithm), but this struggled with our 10-link robot, frequently timing out in all attempts at a query. This is because the FMM\* algorithm is more geared towards lower-dimensional configuration spaces, as explained in [20].

Each algorithm is run with *perturbationRadius* values of 0.5, 1.0, 1.5, and 2.0, corresponding to the *range* parameter for RRT\* and governing the maximum step size taken when extending the tree or connecting sampled configurations. Smaller values yield better exploration and tend to improve success rates in narrow spaces at the cost of longer planning times, whereas larger values promote faster coverage of the configuration space but increase the likelihood of local planner failures through obstacles. Evaluating each planner across these four *perturbationRadius* values enables a systematic characterisation of its sensitivity to step size and provides the empirical basis for identifying combinations that best balance success rate and computational efficiency across different queries in this study.

### B. ML Regression

We use the `RandomForestRegressor` from the `scikit-learn` [21] Python library. The hyperparameters (*criterion*, *max. features*, *max. samples*, *min. samples at leaf*, *# estimators*) are tuned with 20 iterations of randomised 5-fold cross-validation. In total the training takes approximately just under 19 CPU hours (under 2 hours wall time) on 24 cores on one AMD EPYC3 7643 node using 16GB of RAM in total.

### C. Evaluation

For evaluation purposes, we define four *selectors*, i.e. ways of choosing the order and planner:

- **VB Random**: The *virtual best* selector is able to “peek” at the ground truth results for the test set queries, and constructs its sequence by passing to the constraint solver the median time observed for each query for each planner configuration. The sequence of planners

and queries generated by the constraint solver is then simulated by sampling uniformly at random from the 20 times recorded for each query. This selector represents the case when real results are available for the queries between poses in our test set; nevertheless, this selector reflects that, even with this knowledge, the planning algorithms can display unpredictability.

- **VB Lucky**: The sequence of planners and queries is obtained as above, but this time the best (minimum) observed total planning and execution time for each query for the chosen planner is used to calculate the overall time.
- **SB + ML Regression**: From the training set data, the *single best* planner is determined as the one whose mean time across all training queries is the lowest. The constraint solver is then called with a portfolio of size 1 (the single best planner) and is given the ML predictions for total planning and execution time for the chosen solver. Again, the overall time for the resulting plans are calculated by sampling from the times observed for each query.
- **PAMPR (ML + CP)**: The ML regression predicts the combined planning and execution time for each planner and query according to the features extracted. We also implement the (*time + tail*) predictions described earlier. The predicted times are used by the constraint solver to generate a sequence of queries and planners. Once more, the reported time is calculated by sampling from the observed times.

In all cases above, if a plan is not returned in 60 seconds, the resulting time is marked as  $2 \times$  the time limit, i.e. 120 seconds. This penalty allows a fair comparison of overall running times, but we also report on the actual number of timeouts so that we can evaluate the reliability of any given method.

We argue that the single best selector is a reasonable baseline to compare against, representing what has been learnt from historical data in the same context, given that our training and test sets are drawn from the same distribution of poses. We show the two VB selectors to illustrate how close to a perfect or idealistic solution we can get.

### D. RESULTS AND ANALYSIS

The results across our 10 train/test splits are shown in Figure 5 for the setup which uses the (*time + tail*) ML predictions and applies a 2 second switching penalty. The distribution of total times is plotted for all four selectors described above. Because the VB Lucky selector is always able to pick the lowest of the 20 recorded results for each query, the overall time is constant, so it is showed with a vertical dashed line. In test sets 5 and 6 we notice that the SB times are bunched together – this is because the single best planner in fact timed out for two of the queries in each case, whereas the other queries in the set were completed relatively quickly (within 2 seconds each). Across the other sets we mostly observe PAMPR performing considerably faster than the single best. One exception is test set 2 where SB is faster

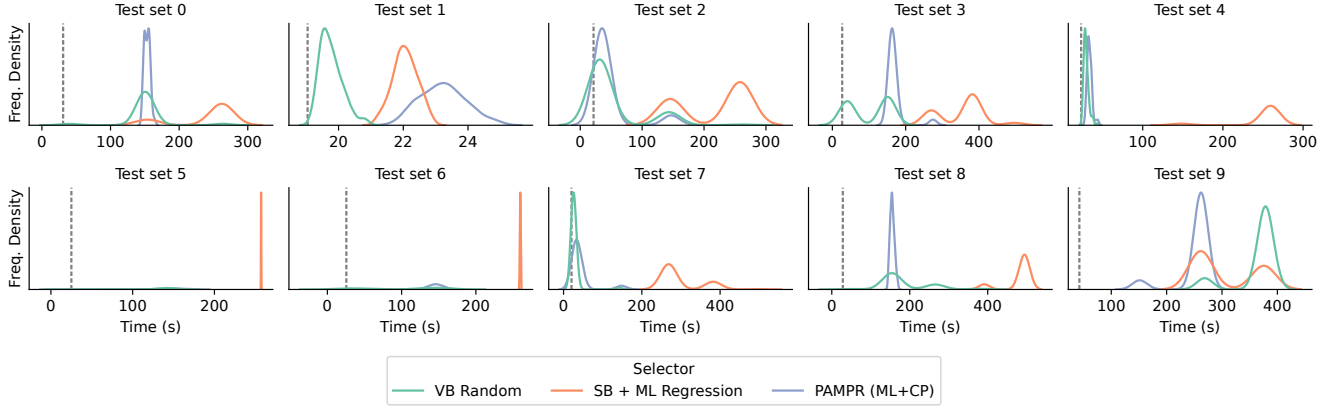


Fig. 5. The distribution of total planning and execution time for the complete tour of poses in 10 test sets, with 100 samples being drawn for each test set. The vertical dashed line indicates the Virtual Best Lucky result. This setup uses the (time + tail) ML predictions and a 2 second switching penalty.

TABLE I

OVERALL PLANNING AND EXECUTION TIME ACROSS 10 TEST SETS

| Setup             | Selector      | Mean   | 95%    | Max.   |
|-------------------|---------------|--------|--------|--------|
| tail, no pen.     | VB Lucky      | 23.95  | 40.57  | 40.57  |
|                   | VB Random     | 108.21 | 374.58 | 383.23 |
|                   | SB + ML Regr. | 266.93 | 492.14 | 499.48 |
|                   | PAMPR (ML+CP) | 110.50 | 259.88 | 281.30 |
| tail, 2 sec. pen. | VB Lucky      | 26.33  | 42.37  | 42.37  |
|                   | VB Random     | 112.93 | 377.43 | 386.37 |
|                   | SB + ML Regr. | 271.88 | 494.16 | 500.10 |
|                   | PAMPR (ML+CP) | 113.17 | 261.79 | 281.44 |
| time, no pen.     | VB Lucky      | 24.01  | 40.51  | 40.51  |
|                   | VB Random     | 108.63 | 374.44 | 383.29 |
|                   | SB + ML Regr. | 270.15 | 492.21 | 498.42 |
|                   | PAMPR (ML+CP) | 122.91 | 269.39 | 380.87 |
| time, 2 sec. pen. | VB Lucky      | 23.96  | 40.48  | 40.48  |
|                   | VB Random     | 110.23 | 374.88 | 383.29 |
|                   | SB + ML Regr. | 268.58 | 492.22 | 498.05 |
|                   | PAMPR (ML+CP) | 124.10 | 269.19 | 391.15 |

TABLE II

SUMMARY OF TIMEOUTS PER STEP

| Setup             | Selector      | Mean | Max. |
|-------------------|---------------|------|------|
| tail, no pen.     | VB Lucky      | 0.00 | 0    |
|                   | VB Random     | 0.70 | 3    |
|                   | SB + ML Regr. | 2.06 | 4    |
|                   | PAMPR (ML+CP) | 0.69 | 2    |
| tail, 2 sec. pen. | VB Lucky      | 0.00 | 0    |
|                   | VB Random     | 0.71 | 3    |
|                   | SB + ML Regr. | 2.09 | 4    |
|                   | PAMPR (ML+CP) | 0.68 | 2    |
| time, no pen.     | VB Lucky      | 0.00 | 0    |
|                   | VB Random     | 0.70 | 3    |
|                   | SB + ML Regr. | 2.09 | 4    |
|                   | PAMPR (ML+CP) | 0.78 | 3    |
| time, 2 sec. pen. | VB Lucky      | 0.00 | 0    |
|                   | VB Random     | 0.72 | 3    |
|                   | SB + ML Regr. | 2.08 | 4    |
|                   | PAMPR (ML+CP) | 0.79 | 3    |

on average than PAMPR; however, here the times are much shorter (less than 30 seconds total), i.e. the queries in this set present a much easier challenge than the other test sets.

In general we observe that compared to the SB, the distribution of times from PAMPR is more consistent with less variability, as well as having a much lower average time in all but one of the 10 test sets. This success on two fronts is further evidenced by the results in Table I. The table shows the mean, 95th percentile and maximum time for all selectors across the 1000 evaluation runs (10 test sets and 100 sampling runs for each plan). The results are shown for the four setups we run, i.e. choosing between the bare time prediction versus the tail-aware prediction, and between no switching penalty and a 2 second penalty for switching planners. We observe that the 95th percentile value is between 45% and 47% faster for PAMPR compared to the single best. Even when we compare the maximum values, we see similar improvements, showing that the tail of the distribution of times is much reduced when using PAMPR. The mean time ranges from 53% faster for the (*time, 2s pen.*)

setup to 58% faster for the (*tail, no pen.*) setup, showing that we do not sacrifice average performance; quite the opposite, PAMPR is also faster on average than the SB.

In Table II we aggregate the number of timeouts per complete tour experienced by each of the selectors across the 1000 simulated executions (recall we have 10 train / test splits and we simulate running the planners 100 times for each test set by sampling from the experimental results). Note that PAMPR is hitting less than one timeout on average per task, whereas the single best is averaging more than 2. It's also worth noting that using the (*time+tail*) predictions is leading to fewer timeouts as these predictions are including a penalty for variability versus the plain *time* predictions which are simply targeting the time itself.

## VI. CONCLUSIONS AND FURTHER WORK

We have presented PAMPR, a framework which uses machine learning to make predictions about the performance of algorithms according to features of the robot and the environment alongside constraint programming to construct a

schedule of algorithms and a sequence of endpoints to reach. We show that this approach leverages the complementarity of different motion planners and configurations in order to produce faster and more robust planning and execution times.

It would be interesting to extend this work in many ways. The features currently used for this proof of concept are obtained as ground truth because we have control of the scene; it would be good to test whether the approach holds up when using a point cloud from sensors in a real physical environment using a physical robot arm. The framework could be applied to other robots and environments, such as a mobile robot in a more complex environment. In algorithm selection literature, the design of the portfolio can be crucial in extracting good performance; we would like to study how a portfolio of planners and hyperparameters can be constructed from training data.

## APPENDIX

### A. Results for all ML Setups

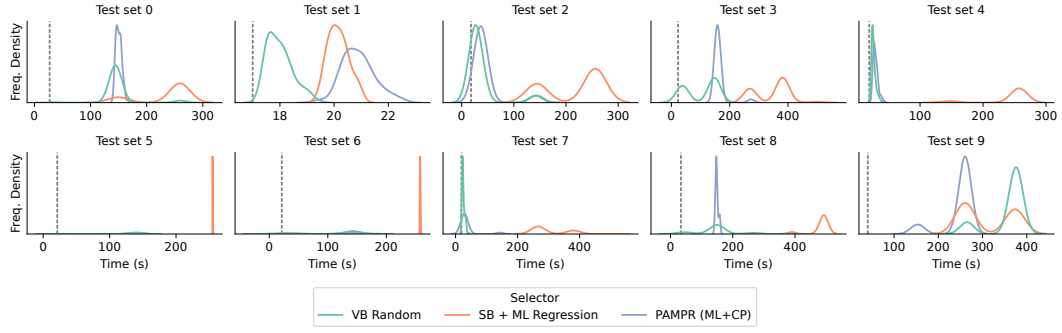
We evaluated four setups for PAMPR, split across two decisions. Firstly, whether to predict the total runtime itself, or the runtime plus the “tail” (50-95th percentile). Secondly, whether to apply a penalty for switching algorithms between steps. In the main paper we showed the plot for the setup which used the (time + tail) prediction and applied a 2 second switching penalty. The distributions of results across the remaining three setups are shown in Figure 6.

## ACKNOWLEDGEMENT

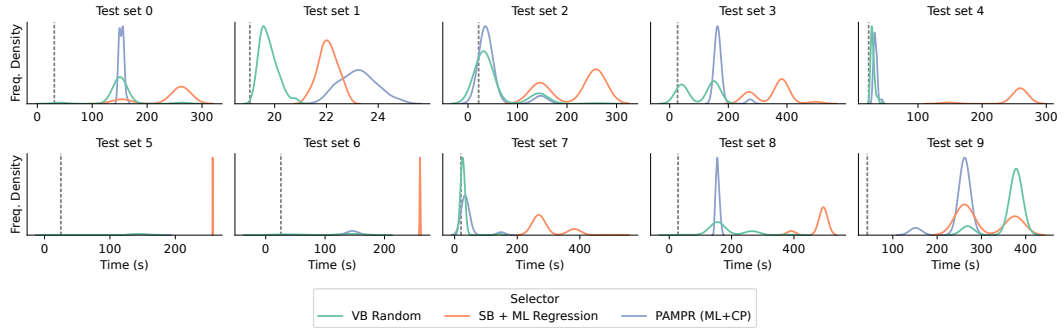
This research was funded in part by Innovate UK SCHEME project (10065634). EPSRC Research Data Management: No new primary data was created during this study. The Viking cluster, a high performance compute facility provided by the University of York, was used for experiments in this work.

## REFERENCES

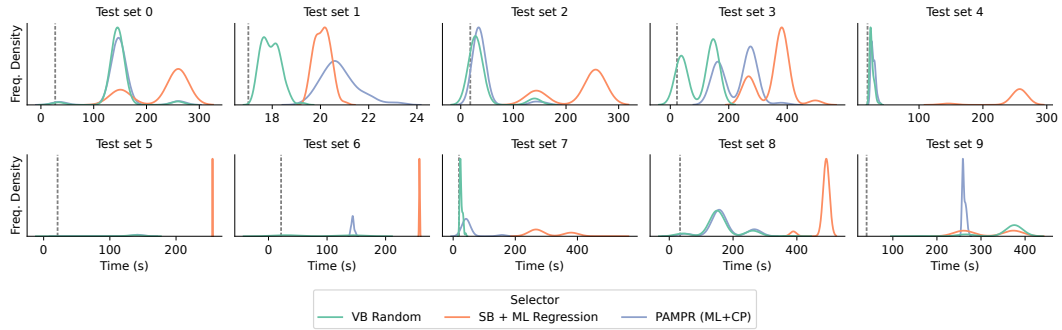
- [1] I. A. Şucan, M. Moll, and L. E. Kavraki, “The Open Motion Planning Library,” *IEEE Robotics & Automation Magazine*, vol. 19, pp. 72–82, December 2012. <https://ompl.kavrakilab.org>.
- [2] S. Yang, P. Liu, and N. Pears, “Benchmarking of Robot Arm Motion Planning in Cluttered Environments,” in *2023 28th International Conference on Automation and Computing (ICAC)*, pp. 1–6, Aug. 2023.
- [3] L. Zhang, K. Cai, Z. Sun, Z. Bing, C. Wang, L. Figueredo, S. Haddadin, and A. Knoll, “Motion planning for robotics: A review for sampling-based planners,” *Biomimetic Intelligence and Robotics*, vol. 5, p. 100207, Mar. 2025.
- [4] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham, “Mode change protocols for priority-driven preemptive scheduling,” *Real-Time Systems*, vol. 1, pp. 243–264, Dec. 1989.
- [5] J. R. Rice, “The Algorithm Selection Problem,” in *Advances in Computers* (M. Rubinoff and M. C. Yovits, eds.), vol. 15, pp. 65–118, Elsevier, Jan. 1976.
- [6] P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann, “Automated Algorithm Selection: Survey and Perspectives,” *Evolutionary Computation*, vol. 27, pp. 3–45, Mar. 2019.
- [7] J. Jeon, H.-r. Jung, T. Luong, F. Yumbla, and H. Moon, “Combined task and motion planning system for the service robot using hierarchical action decomposition,” *Intelligent Service Robotics*, vol. 15, pp. 487–501, Sept. 2022.
- [8] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, “Integrated Task and Motion Planning,” *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, pp. 265–293, May 2021.
- [9] A. Sripathy, A. Bobu, D. S. Brown, and A. D. Dragan, “Dynamically Switching Human Prediction Models for Efficient Planning,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3495–3501, May 2021.
- [10] P. Nightingale, Ö. Akgün, I. P. Gent, C. Jefferson, I. Miguel, and P. Spracklen, “Automatically improving constraint models in Savile Row,” *Artificial Intelligence*, vol. 251, pp. 35–61, Oct. 2017.
- [11] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack, “MiniZinc: Towards a Standard CP Modelling Language,” in *Principles and Practice of Constraint Programming – CP 2007* (C. Bessière, ed.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 529–543, Springer, 2007.
- [12] L. Perron and F. Didier, “CP-SAT,” [https://developers.google.com/optimization/cp/cp\\_solver/](https://developers.google.com/optimization/cp/cp_solver/).
- [13] L. Breiman, “Random Forests,” *Machine Learning*, vol. 45, pp. 5–32, Oct. 2001.
- [14] K. Hauser, “Robust Contact Generation for Robot Simulation with Unstructured Meshes,” in *Robotics Research: The 16th International Symposium ISRR* (M. Inaba and P. Corke, eds.), pp. 357–373, Cham: Springer International Publishing, 2016.
- [15] “Klampa’t software suite,” <https://motion.cs.illinois.edu/klampt/>. Accessed: 2026-03-19.
- [16] Kinova Robotics, “Jaco robot arm,” <https://assistive.kinovarobotics.com/product/jaco-robotic-arm>. Accessed: 2026-04-21.
- [17] G. Sánchez and J.-C. Latombe, “A Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking,” in *Robotics Research* (R. A. Jarvis and A. Zelinsky, eds.), (Berlin, Heidelberg), pp. 403–417, Springer, 2003.
- [18] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *The International Journal of Robotics Research*, vol. 30, pp. 846–894, June 2011.
- [19] K. Hauser, “Lazy collision checking in asymptotically-optimal motion planning,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, (Seattle, WA, USA), pp. 2951–2957, IEEE, May 2015.
- [20] J. Luo and K. Hauser, “An empirical study of optimal motion planning,” in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Chicago, IL, USA), pp. 1761–1768, IEEE, Sept. 2014.
- [21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.



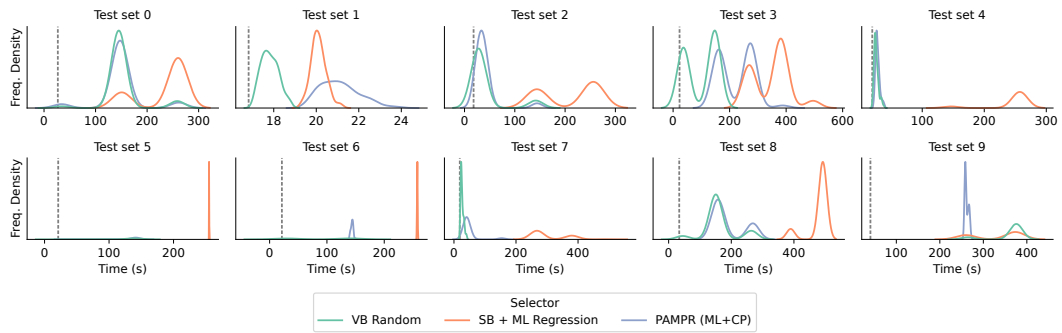
(a) Time + tail, no penalty



(b) Time + tail, no penalty



(c) Time + tail, no penalty



(d) Time + tail, no penalty

Fig. 6. Planning and execution time over 10 test sets using different ML prediction targets (time only vs. time + tail) and different algorithm switching penalties (no penalty vs. 2 seconds)