GRAPH GAUSSIAN PROCESSES FOR EFFICIENT ROBUST MONTE CARLO TREE SEARCH

Dorina Weichert
Fraunhofer Institute for Intelligent Analysis and Information Systems IAIS
53757 Sankt Augustin
Germany

Samuel Wiest & Sebastian Houben & Paul Ploeger
University of Applied Sciences Bonn-Rhein-Sieg
53757 Sankt Augustin
Germany

ABSTRACT

One of the major challenges in applying machine learning-based optimization algorithms in practice is their efficiency, which is measured in runtime and sample efficiency. Unfortunately, these two measures do not go hand in hand, but there are two poles: model-based Reinforcement Learning (RL), which is fast but requires many calls to some oracle, and Bayesian Optimization (BO), which has a high computation time but is very sample efficient. Additionally, both methods are at risk of oracle-misspecification: the oracle used for learning may differ from the final application. We derive Graph Gaussian Process Monte Carlo Tree Search (GUMTREES), an algorithm combining Monte Carlo Tree Search (MCTS), a model-based RL method, with BO, leading to an efficient algorithm that is easily enhanced to the robust setting. In a simple experiment, we demonstrate the superior performance of our algorithm.

1 INTRODUCTION

The practical application of methods from RL and BO is impeded by efficiency constraints. While model-based RL in particular is often fast in training, it requires a high number of calls to an oracle or even a real environment O; BO has a high runtime but is very sample efficient. We introduce an approach combining the best of both worlds: GUMTREES. To do so, we combine ideas from MCTS (please see Browne et al. (2012) for a review) with Combinatorial Bayesian Optimization (COMBO) (Oh et al., 2019), resulting in an algorithm with (reasonable) tradeoff between sample-efficiency and runtime. Additionally, we treat one of the most critical risks when applying RL in real-world settings: the risk of a misspecified oracle O during training. This might even happen for model-free methods like BO when the optimization process takes place in an environment different from the final application or when facing concept drift. We implement this challenge by considering a set of oracles $\mathcal{O}$ instead of a single one. We evaluate our approach on a basic experiment, showing its superior behavior in both efficiency and robustness against adversarial oracle definitions.

1.1 COMBO ALGORITHM

Graph Gaussian Processes Gaussian Processes (GP) Regression is a non-parametric method to model an unknown function $f(\mathbf{x}) : \mathcal{X} \mapsto \mathbb{R}$ by a distribution over functions. The GP prior is defined such that any subset of function values is normally distributed with mean $\mu_0(\mathbf{x})$ and covariance $k(\mathbf{x}, \mathbf{x}')$ for any $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ (w.l.o.g. we assume $\mu_0(\mathbf{x}) = 0$, see e.g. Rasmussen & Williams (2006)). Conditioning the prior on actual data $D_t = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_t, y_t)\}$, where $\mathbf{y} = f(\mathbf{x}) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$, the predictive posterior distribution $p(f) \sim GP(m, v|D_t)$ is given by $m(\mathbf{x}|D_t) = \mathbf{k}(\mathbf{x})^T \mathbf{K}^{-1} \mathbf{y}$, $v(\mathbf{x}|D_t) = k(\mathbf{x}, \mathbf{x}) - \mathbf{k}(\mathbf{x})^T \mathbf{K}^{-1} \mathbf{k}(\mathbf{x})$, with $[\mathbf{k}(\mathbf{x})]_i = k(\mathbf{x}, \mathbf{x}_i)$, $\mathbf{K}_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j) + \delta_{ij}\sigma_n^2$, where δ_{ij} is the Kronecker delta, and $[\mathbf{y}]_i = y_i$.

Typically, covariance functions are defined on continuous spaces. For combinatorial graphs $G = (V, E)$, with vertices V and edges E, Oh et al. (2019) define a covariance function to model a function f on the vertices V as a Graph Gaussian Process (GRAPH GP). Given N complete subgraphs $\{G(C_i)\}$ for categorical variables $C_1, \ldots, C_N$, the full combinatorial graph is built via the graph Cartesian product $\Box_i G(C_i)$. By Graph Fourier Transform (GFT), a smooth function on a graph is defined using a linear combination of graph Fourier bases. Calculation of the GFT is performed via the graph Laplacian $L(G)$. Its eigenvalues $\{\lambda_1, \lambda_2, \ldots, \lambda_{|V|}\}$ and eigenvectors $\{u_1, u_2, \ldots, u_{|V|}\}$ are the graph Fourier frequencies and bases. To penalize high (and therefore expected less relevant) frequencies, a diffusion kernel is used: For vertices p and q, the kernel is defined by $k(p, q) = \sum_{i=1}^{n} e^{-\beta\lambda_i} u_i(p) u_i(q)$, where β is an additional hyperparameter: the lengthscale. Unfortunately, calculating the graph Laplacian is infeasible for large graphs. For combinatorial graphs, there is significant simplification by the graph Cartesian product, using the Kronecker product $\otimes$ and the Kronecker sum $\oplus$. For c categorical variables Oh et al. (2019) find: $\mathbf{K} = \exp\left(-\beta \oplus_{i=1}^{c} L\left(G(C_i)\right)\right) = \otimes_{i=1}^{c} \exp\left(-\beta L\left(G(C_i)\right)\right)$, making the calculation of the GFT only necessary for the smaller subgraphs.

Learning a GRAPH GP is computationally heavy. The original authors propose to use slice sampling to estimate the necessary hyperparameters β and σ_n Oh et al. (2019), which, despite being a powerful method in contrast to traditional maximization of the marginal log-likelihood, leads to a high runtime of the algorithm.

Solving Markov Decision Process (MDP)s by the COMBO algorithm Basic combinatorial BO with GRAPH GPs is performed analogously to traditional BO: After definition of the search space, consisting of the full combinatorial graph defined by the individual subgraphs for every categorical variable, an acquisition function, such as Expected Improvement (EI) is applied to find the next vertex V for evaluation. Given the corresponding response $\mathbf{y}_t$ through the environment O, the GRAPH GP is updated and the acquisition function is called again. This procedure is repeated until some terminal condition is met and an optimal combination of variables is found. In general, BO is known to be very sample efficient on the cost of a long runtime (Shahriari et al., 2016; Garnett, 2023).

COMBO is also suitable to solve simple MDPs with discrete action spaces $\mathcal{A}$. After choosing a maximum sequence length T, a combinatorial graph is generated from T identical subgraphs $C_1, \ldots, C_T$ whose vertices V correspond to the individual actions $a \in \mathcal{A}$. Given this combinatorial graph that represents every possible combination of actions and their order, the cumulative reward $R_T(V)$ is modeled by the GRAPH GP. BO is then used to find an optimal sequence of actions.

1.2 MCTS ALGORITHM

MCTS is a RL method for MDPs with large state spaces $\mathcal{S}$ that heavily uses an oracle $(s', r) \sim O(s, a)$. The following introduction builds on the review by Browne et al. (2012). Basically, the goal of MCTS is to find an optimal sequence of T actions maximizing the cumulative reward $R_T = \sum_{t=0}^{T} r_t$. It builds a search tree, where each node of the search tree corresponds to a state s and each edge to an action a. The tree is iteratively expanded from the root node that represents an initial state s_0. Each iteration consists of four main steps: selection, expansion, simulation, and backpropagation.

Selection is performed via a so-called tree policy, which is similar to an acquisition function in BO, trading off exploration and exploitation. Beginning from the root node, the tree policy is calculated for all possible actions from the actual state, and the one with the maximum value of the tree policy is chosen. Via the oracle, the subsequent state and the immediate reward $(s', r) \sim O(s, a)$ are calculated and $s' = s$ is set. This selection step is repeated until the actual node has unvisited children, referring to actions that have yet to be explored. In the *expansion* step, one action from the unvisited ones is randomly chosen and executed, and the result is used to extend the tree, again making use of the oracle. Afterward, *simulation* follows: a default policy, typically a random action selection up to a specific horizon, is used to estimate the cumulative reward corresponding to the new node. Finally, the gained value is *backpropagated* to the tree's root, i.e., adding all generated future rewards on the path to the finally extended node to its parent node. This procedure is repeated until a terminal criterion, usually a specific budget of iterations, is met.

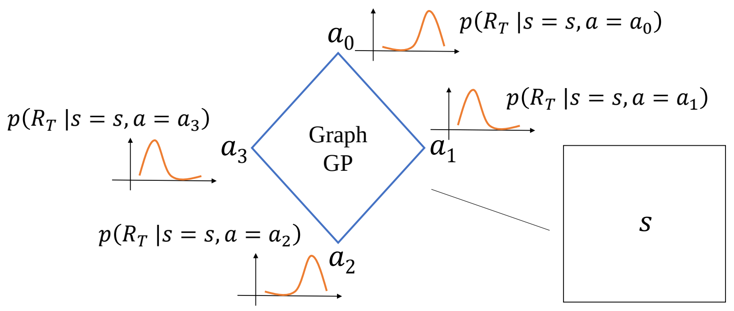

$p(R_T \mid s = s, a = a_0)$
a_0
$p(R_T \mid s = s, a = a_1)$
a_1
$p(R_T \mid s = s, a = a_3)$
a_3
Graph GP
$p(R_T \mid s = s, a = a_2)$
a_2
s

Figure 1: Basic idea of GUMTREES. We model the cumulative reward R_T for each state s for each action $a \in \mathcal{A}$ by a GRAPH GP. In the figure, we show a single state with four actions.

Afterward, the following action a to take on the real environment (which might be the cheap oracle, but does not have to) is selected by some criterion, such as choosing the action with the highest cumulative reward. In contrast to solving an MDP with COMBO, this is an online approach, meaning that it does not return an entire action sequence but is short-sighted, considering the uncertainty of long action sequences.

2 GUMTREES ALGORITHM FOR ROBUST SAMPLE-EFFICIENT RL

2.1 GENERAL METHOD

Our idea in GUMTREES is to combine sample-efficient BO with fast MCTS to generate a sample-efficient and fast solver for MDPs that has the additional advantage of easily taking into account model uncertainty. For each state s, we generate a GRAPH GP to model the distribution over cumulative rewards $p(R_T|s, a)$ when taking out a sequence of actions that begins with the action a, so we assume $p_s(R_T) \sim GP(m_s, v_s)$. Figure 1 emphasizes this basic idea for a single state s_0 and four actions $a_0, \ldots, a_3$. For each vertex corresponding to an action a, we show the predictive distribution $p(R_T|s, a)$, which is individual for each action a. Please note that this predictive distribution exists even if an action a has yet to be visited. In contrast to other RL methods, such as MCTS, we work with a distribution over cumulative rewards instead of using some scalar measure. Assuming a relatively small state space, we generate as many GRAPH GPs as states.

Searching for an optimal sequence of actions, we adopt the general idea of MCTS to search sequentially in a "selection, expansion, simulation and backpropagation"-manner, beginning from some initial state s_0. Using the predictive distribution over cumulative rewards $p(R_T)$, we *select* the next action a to take. Here, all kinds of acquisition functions, such as EI, Thompson Sampling (TS), or Upper Confidence Bounds, are possible. Using the oracle O, we generate the next state $s' = O(s, a)$. If the subsequent state s' has not been visited before, we generate a new GRAPH GP (similar to the *expansion* step in MCTS) and *simulate* via random action selection until some terminal state, associated with some cumulative return R_T is reached. Finally, we *propagate back* the value of R_T, update the cumulative reward for all vertices of GRAPH GPs in the search trajectory, and retrain them.

After reaching a budget of evaluations B, we select the following action to take from the set of GRAPH GPs $\mathcal{G}$. Therefore, we extract the graph corresponding to the initial state s_0 and find the action with the highest predictive mean cumulative reward m_{s_0}: $a = \arg\max_{a \in \mathcal{A}} m_{s_0}(a)$. We summarize the described steps in algorithm 1.

Even though the method is similar to MCTS, there is one potential advantage of our approach. As we model the cumulative rewards, we take into account their correlation via the lengthscale β. Therefore, we can distinguish between states where the exact selected action is highly relevant, i.e., close to the terminal state, or states with room for freedom in which action to take, i.e., initial actions. Therefore, we expect fewer calls to the oracle O than with MCTS.

Algorithm 1 GUMTREES algorithm.

```
Input  oracle $O$, action space $\mathcal{A}$, evaluation budget $B$, initial state $s_0$
Output  optimal action $a$
 1: $\mathcal{G} \leftarrow \emptyset, R_T \leftarrow 0, s \leftarrow s_0$
 2: for $b = 1, \ldots, B$ do
 3:     if $g(s) \in \mathcal{G}$ then
 4:         $a \leftarrow$ SELECTACTION($g(s)$)          ▷ Select action using an acquisition function.
 5:         $s', r \leftarrow O(s, a)$
 6:         $s \leftarrow s', R_T \leftarrow R_T + r$
 7:     else
 8:         if NOTTERMINAL($s$) then                    ▷ Check if state $s$ is terminal.
 9:             $g(s) \leftarrow$ GENERATEGGP($s$)        ▷ Generate new GRAPH GP.
10:             $\mathcal{G} \leftarrow \mathcal{G} \cup g(s)$
11:             $R_T \leftarrow R_T +$ SIMULATE($s$)      ▷ Simulate the future rewards.
12:             BACKUP($R_T$), $s \leftarrow s_0$         ▷ Backup and restart from initial state.
13:         else
14:             BACKUP($R_T$), $s \leftarrow s_0$         ▷ Backup and restart from initial state.
15:         end if
16:     end if
17: end for
18: return BESTACTION($g(s_0)$)                         ▷ Select next action to take.
```

2.1.1 Dealing with Multiple Evaluations of Vertices

Critical for our approach is the multiple evaluation of vertices with (probably) different returns. This is a problem for GPs, as the upcoming covariance matrix is not positive semidefinite anymore due to duplicate entries, and no predictions can be made. However, multiple evaluations at the same location also contain relevant information about the noise variance of the problem that is represented by the covariance function's hyperparameter σ_n. As σ_n impacts the posterior predictive distribution $GP(m, v)$, it influences the explorative behavior of the algorithm.

In the following, we describe three options to deal with this problem. While the first one, the use of the mean cumulative reward for the multiple evaluations, is homoscedastic, the others (a heuristic and a Bayesian approach) are not.

Application of Mean Cumulative Reward The simplest approach to treat multiple evaluations at the same vertex is to apply the mean cumulative reward of the different evaluations. In the experiments, this approach will be called "Mean R_T". Despite its attractiveness due to its simplicity, this approach has several drawbacks: First, it does not at all use the available information about the noise variance, so the value found by fitting the GRAPH GP model might be far from the actual value. Second, when using the defaults from COMBO's GRAPH GPs, the model is homoscedastic, meaning that the same noise parameter σ_n is applied for all vertices. Even though this assumption might be valid for some states s, it is at least problematic close to a terminal state s, where one particular action leads to the end of the action sequence. We also tried to apply heteroscedastic $\sigma_{n,V}$ for each vertex. However, we could not run experiments with that approach due to the computationally heavy slice sampling procedure of the hyperparameters.

Heuristic Approach Recognizing the relevance of assuming heteroscedasticity but trying to reduce computation time, we propose the following heuristic, relying on the number of evaluations of the individual vertex N_{V_s} and the depth of the search path d_s to the associated state s: $\sigma_{n_V} = \frac{1}{d_s \cdot N_{V_s}}$. The basic ideas behind the heuristic are the following: the more often a vertex V_s is visited, the lower the noise variance of the cumulative reward R_T, especially as the samples are drawn non-iid, but in favor of finding a high cumulative reward R_T. Following the same rationale, the deeper the search path d_s to the associated state s, the higher the probability of being close to a terminal state. Even though this approach is relatively simple and additionally leads to a vanishing noise variance σ_{n_V} for a high depth and/or number of evaluations, instead of a constant value, experiments show promising results. This approach will be called "Heuristic" in the experiments.

Bayesian Approach In the spirit of the Bayesian background of GRAPH GPs in general, we fit the individual vertices noise hyperparameter σ_{n_V} using conjugate priors. Conjugate priors are algebraically convenient, allowing for an analytic expression of the posterior distribution for a given likelihood. Assuming σ_{n_V} to be normally distributed, we apply a Normal-inverse gamma distribution as conjugate prior. In preliminary tests, we found a disastrous behavior of this approach, which could be fixed by changing the simulation step of the GUMTREES algorithm from random action selection to best case action selection, meaning to simulate the trajectory leading to a positive terminal state. In the experiments, this approach will be called "Bayesian".

2.2 ROBUST ADAPTION OF ALGORITHM

Given the general idea of GUMTREES, it is easy to extend the algorithm to match our robustness requirements, meaning to find the optimal action facing a finite set of possible oracles $\mathcal{O}$. We define the term optimal in a conservative way as being the action leading to the highest cumulative future reward even under the worst-case assumed model, so $a^\star = \arg\max_{a \in \mathcal{A}} \min_{O \in \mathcal{O}} R_T(s, a, O)$.

We leverage one of the advantages of COMBO: the simple augmentation of categorical variables via subgraphs and add a fully connected subgraph where each vertex corresponds to one of the possible oracles. This generates an entire graph, where each vertex V corresponds to an action-oracle tuple (a, O). Using the covariance function with a two-dimensional lengthscale hyperparameter $\boldsymbol{\beta} = [\beta_a, \beta_o]$, where β_a is associated with the actions and β_o with the oracles, the GRAPH GPs represent cumulative rewards $R_T(s, a, O)$ specifically for these parameters and a given state s. Advantageously, it can also learn the similarity of the models and actions via $\boldsymbol{\beta}$: therefore, not all combinations have to be visited to create a good prediction. Additionally, the selection step of GUMTREES is adapted to match the multi-oracle setting: instead of only selecting an action via a standard acquisition function, we take into account the robustness requirement and select an action oracle tuple $(a^\star, O^\star)$. Therefore, we use simple TS, meaning that we draw one sample $\hat{R}_T(s, a, O)$ from the predictive distributions at each vertex and select via a min-max criterion:

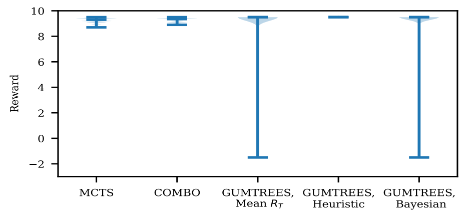
$$(a^\star, O^\star) = \arg\max_{a \in \mathcal{A}} \arg\min_{O \in \mathcal{O}} \hat{R}_T(s, a, O) \ . \tag{1}$$

Alternatively, other robust acquisition functions, such as the variations by Bogunovic et al. (2018) or Weichert & Kister (2020), could be used.

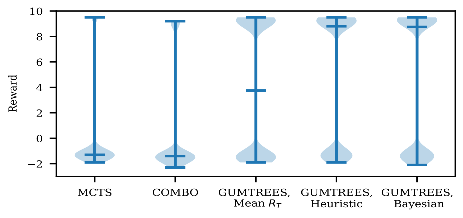
3 EXPERIMENTS

We benchmark our approach on a simple experimental setup against the COMBO algorithm and MCTS: the frozen lake environment by Gymnasium (Towers et al., 2023). To compare the efficiency of our algorithms, we limit the number of calls to the oracles $\mathcal{O}$ and analyze the reached cumulative reward R_T. We use 270 iterations for COMBO, which is the default and refers to the maximum number of oracle calls being 270 times the mean problem-specific sequence length $T = 15$, which allows for exploration. We show two metrics: the runtime of the individual algorithms for training and application and their performance measured in terms of cumulative reward R_T when applying the resulting actions gained by the algorithms to the different environments. Due to the high runtimes, especially of COMBO, we take a shortcut and directly evaluate an entire action sequence instead of alternating between calculating the best action from the current state s and applying it on an environment, leading to a new state s'. Therefore, we extract the best sequence of actions from MCTS and GUMTREES instead of a single action, assuming that the search tree built by the oracle indeed corresponds to the behavior of the environment. As we are mainly interested in the robustness characteristics of GUMTREES, we use an experimental setup with two oracles $\mathcal{O} = \{O_1, O_2\}$. We train GUMTREES on both oracles simultaneously, while MCTS and COMBO are trained on each oracle individually. We then apply the results separately to the oracles so we can analyze the algorithms' behavior when trained and applied with and without adversarial oracles. Results for training on a single oracle $\mathcal{O}$ are given in appendix A.2.

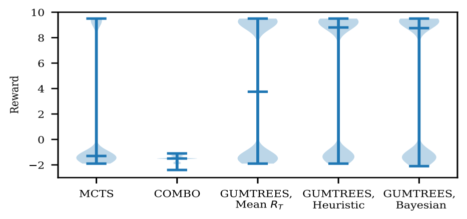
Frozen Lake Experiment The frozen lake environment (Towers et al., 2023) is a simple environment where an agent has to find their way over a frozen lake with holes. There are 16 possible states s, including the initial and the terminal ones, and 4 possible actions $\mathcal{A} = \{a_1, a_2, a_3, a_4\}$ indicating if the agent is moving up, down, left or right. There exist two versions of the environment: a



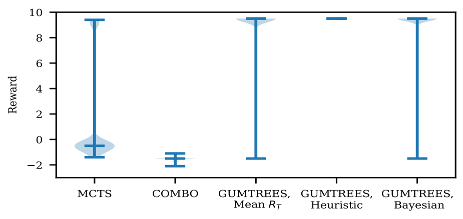
(a) Training and testing on O_1.



(b) Training on O_1, testing on O_2.



(c) Training and testing on O_2.



(d) Training on O_2, testing on O_1.

Figure 2: Rewards for all algorithms for different combinations of oracles for training and testing. GUMTREES always uses $\mathcal{O} = \{O_1, O_2\}$ for training. The horizontal line indicates the median.

non-slippery one O_1 with deterministic actions, meaning that executing a specific action a from a given state s leads to only one subsequent state s'. Alternatively, there is the slippery environment O_2, where the behavior of the actions is stochastic, meaning that stepping in a specific direction is associated with a probability $\hat{p}$ of moving in an orthogonal direction, e.g., if action a_2, associated with moving down is taken, there is a certain probability of instead moving left or right. We set $\hat{p}$ to the default of 0.3. Each action a is associated with an immediate reward $r_t = -0.1$, falling into a hole is associated with a reward $r_t = -1$, and finding the positive terminal state is associated with a reward $r_t = 10$. In our version (please see figure 3 in appendix), there are two possible paths to the terminal state where one entails more risk. The experiment's goal is to test whether our robust version of GUMTREES can find the less risky, lower path.

Figure 2 shows the performance results. While all algorithms perform well for the case of training and testing on deterministic O_1, MCTS and COMBO fail for training on O_1 and testing on stochastic O_2 in most of the cases. This result is expected: while learning in O_1 will return the less risky path with a chance of 50%, its application may still lead to slipping in one of the holes, corresponding to a negative terminal state. In contrast, GUMTREES seems to find the less risky path during training on $\mathcal{O} = \{O_1, O_2\}$, as the results for application on O_2 indicate. The worse performance of the Mean R_T-version is due to its homoscedasticity assumption. Unfortunately, MCTS seldom achieves the positive terminal state when training on stochastic O_2, COMBO never. For COMBO, this is due to the fact that it is not designed to work with stochastic oracles - an action sequence that works during training may not work at application time. As COMBO does not revisit vertices, i.e., action sequences, the algorithm is unaware of that effect. For MCTS, there is one main reason for its bad performance: the limitation of calls to the oracle. As the search tree becomes very broad, as calling an action may lead to three different states, the number of visits in the later nodes becomes small. Therefore, subsequent shares of the finally selected action sequence are unstable.

In table 1, we report the average computing time of the algorithm for the case of training and testing on O_1. As expected, the runtime of GUMTREES is between that of COMBO and MCTS. Similar results for other combinations of oracles are given in appendix A.2.1.

4 Conclusion and Future Work

We introduced GUMTREES, an algorithm for efficient and robust RL. It combines ideas from fast MCTS with sample-efficient BO to find an algorithm that is also easily adapted to face the problem of multi-oracle setups. In future work, we would like to further validate our approach using more sophisticated real-life and large-scale experiments. Additionally, we like to adopt GUMTREES to

Table 1: Average computation time per trial for testing on O_1. Training on O_1 for MCTS and COMBO. Training of GUMTREES on $\{O_1, O_2\}$. We report the mean $\pm$ standard deviation across 30 trials. All units are in seconds. Timing experiments was performed on 4 cores of a Intel(R) Xeon(R) CPU E5-2667 v4.

MCTS	**COMBO**	**GUMTREES, Mean R_T**	**GUMTREES, Heuristic**	**GUMTREES, Bayesian**
0.53 ± 0.08	7228.57 ± 37.00	4977.59 ± 228.55	3909.31 ± 108.74	3805.17 ± 116.45

a more tree-like structure: instead of building on a set of graphs G that are associated with states s, we like to associate the graphs with action sequences, similar to the tree in MCTS. We expect this approach to enhance the scalability for large state-spaces, especially in stochastic settings.

Author Contributions

Dorina Weichert developed the approach and the test setup. Additionally, she wrote the first draft of the manuscript. Samuel Wiest implemented the approach and performed the experiments. Sebastian Houben and Paul Ploeger helped with revising the manuscript.

Acknowledgments

This research has been partly funded by the Federal Ministry of Education and Research of Germany and the state of North Rhine-Westphalia as part of the Lamarr Institute for Machine Learning and Artificial Intelligence, Sankt Augustin, Germany, and partly developed in the Fraunhofer Lighthouse Project "SWAP".

References

I. Bogunovic, J. Scarlett, S. Jegelka, and V. Cevher. Adversarially robust optimization with gaussian processes. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. P. Liebana, S. Samothrakis, and S. Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions Computational Intelligence and AI in Games*, 4:1–43, 2012.

R. Garnett. *Bayesian optimization*. Cambridge University Press, 2023.

C. Oh, J. Tomczak, E. Gavves, and M. Welling. Combinatorial bayesian optimization using the graph cartesian product. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*. MIT Press, 2006.

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*, 104(1):148–175, 2016.

M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. de Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis. Gymnasium, 2023.

D. Weichert and A. Kister. Bayesian optimization for min max optimization. In *Workshop on Real World Experiment Design and Active Learning at International Conference on Machine Learning*, 2020.

A Appendix

A.1 Illustration of Frozen Lake Problem

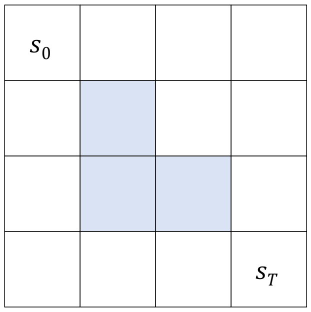


Figure 3: Illustration of the used frozen lake problem. There are two optional paths from state s_0 to the terminal state s_T, with the one through the upper right corner being less risky in the case of the slippery model than the path through the lower left corner. s_0: initial state, s_T: terminal state.

A.2 Use of GUMTREES on a Single Oracle

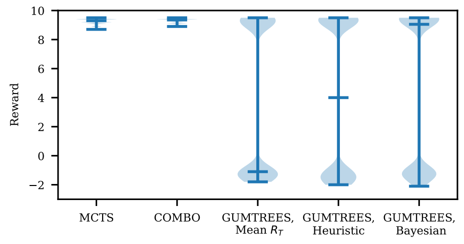
The following shows the results for applying GUMTREES on a single oracle. We use the same experiment as in the main paper; the only difference is that GUMTREES does not have access to two oracles during training.

The runtimes, reported in table 2, change only slightly.

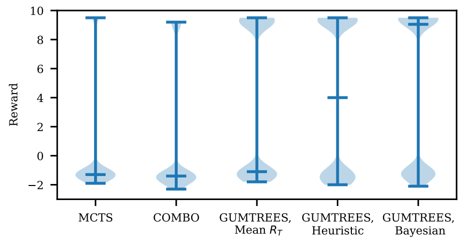
The performance results are given in figure 4. Overall, results become worse for all cases: the possibility to call a slightly different model seems to help learning. Even though GUMTREES still has a superior performance over MCTS and COMBO on mixed train-test settings, e.g., training on O_1 and testing on O_2 and vice versa, its results are mixed for training and testing on the deterministic environment. A hypothesis explaining this result lies in the heuristic behavior of the environment - GPs usually require a certain amount of noise for a good fit - this is not provided for all actions a leading to a terminal state. This idea is supported by the achieved median cumulative rewards, which differ for all approaches; the Bayesian version has an advantage, as it combines our assumptions in the prior with the data gained by exploration. We also find worse results for training on the stochastic oracle O_2 than training on two oracles. Here, the gain of a deterministic model to find an optimal path is confirmed. Despite its disadvantages for purely deterministic applications, GUMTREES is a good alternative for stochastic applications or those in which the exact environment is not known.

Table 2: Average computation time per trial of different algorithms for single-oracle training. We report the mean ± standard deviation across 30 trials. All units are in seconds. Timing experiments was performed on 4 cores of a Intel(R) Xeon(R) CPU E5-2667 v4.

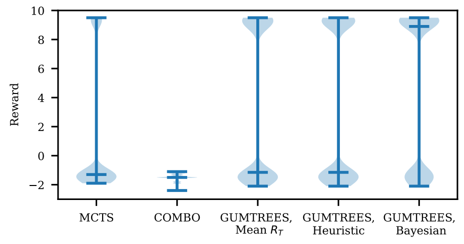
Algorithm	**Train O_1, test O_1**	**Train O_1, test O_2**	**Train O_2, test O_2**	**Train O_2, test O_1**
MCTS	0.53±0.08	0.58±0.11	0.52±0.13	0.51±0.11
COMBO	7228.57±37.00	7226.30±50.42	8634.10±1462.24	8408.63±1267.10
GUMTREES, Mean R_T	4212.26±153.76	4212.26±153.76	4864.00±249.58	4864.00±249.58
GUMTREES, Heuristic	3124.76±26.14	3124.76±26.14	3640.41±114.07	3640.41±114.07
GUMTREES, Bayesian	3183.55±39.91	3183.55±39.91	3823.08±249.58	3823.08±249.58



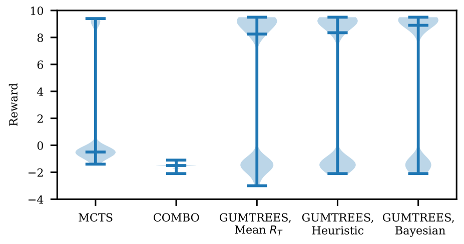
(a) Training and testing on O_1.



(b) Training on O_1, testing on O_2.



(c) Training and testing on O_2.



(d) Training on O_2, testing on O_1.

Figure 4: Rewards for nonrobust training of GUMTREES. The horizontal line indicates the median.

A.2.1 Runtime Results for Experiment in the Main Paper.

Table 3: Extended average computation time per trial of different algorithms for experiment in the main paper. We report the mean ± standard deviation across 30 trials. All units are in seconds. Timing experiments was performed on 4 cores of a Intel(R) Xeon(R) CPU E5-2667 v4.

Algorithm	**Train O_1, test O_1**	**Train O_1, test O_2**	**Train O_2, test O_2**	**Train O_2, test O_1**
MCTS	0.53±0.08	0.58±0.11	0.52±0.13	0.51±0.11
COMBO	7228.57±37.00	7226.30±50.42	8634.10±1462.24	8408.63±1267.10
GUMTREES, Mean R_T	4977.59±228.55	4968.41±232.81	4968.41±232.81	4977.59±228.55
GUMTREES, Heuristic	3909.31±108.74	3851.87±196.86	3851.87±196.86	3909.31±108.74
GUMTREES, Bayesian	3805.17±116.45	3944.72±232.81	3944.72±232.81	3805.17±116.45