TRUST, BUT VERIFY: MODEL-BASED EXPLORATION IN SPARSE REWARD ENVIRONMENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose *trust-but-verify* (TBV) mechanism, a new method which uses model uncertainty estimates to guide exploration. The mechanism augments graph search planning algorithms with the capacity to deal with learned model's imperfections. We identify certain type of frequent model errors, which we dub *false loops*, and which are particularly dangerous for graph search algorithms in discrete environments. These errors impose falsely pessimistic expectations and thus hinder exploration. We confirm this experimentally and show that TBV can effectively alleviate them. TBV combined with MCTS or Best First Search forms an effective model-based reinforcement learning solution, which is able to robustly solve sparse reward problems.

1 INTRODUCTION

Model-based approach to Reinforcement Learning (RL) brings a promise of data efficiency, and with it much greater generality. However, it is still largely an open question of how to make robust model-based RL algorithms. In most cases, the current solutions excel in low sample regime but underperform asymptotically, see Wang et al. (2019); Nagabandi et al. (2018a); Kaiser et al. (2020). The principal issues are the imperfections of the learned model and fragile planners not able to robustly deal with these imperfections, see (Sutton & Barto, 2018, Section 8.3), (François-Lavet et al., 2018, Section 6.2), (Wang et al., 2019, Section 4.5).

Model errors are unavoidable in any realistic RL scenario and thus need to be taken into account, particularly when planning is involved. They can be classified into two categories: optimistic and pessimistic, see (Sutton & Barto, 2018, Section 8.3). The former is rather benign or in some cases even beneficial, as it can boost exploration by sending the agent into falsely attractive areas. It has a self-correcting mechanism, since the newly collected data will improve the model. In contrast, when the model has a pessimistic view on a state, the agent might never have the incentive to visit it, and consequently, to make the appropriate adjustments. In this work, we propose *trust-but-verify* (TBV) mechanism, a new method to augment planners with capacity to prioritise visits in states for which the model is suspected to be pessimistic. The method is based on uncertainty estimates and is agnostic to the planner choice.

Using the graph structure of the underlying problem might be beneficial during planning but also makes it more vulnerable to model errors. We argue that graph search planners might benefit most from utilising TBV. While TBV can correct for any type of pessimist errors, we explicitly identify certain type of model errors which frequently occur in discrete environments, which we dub *false loops*. In a nutshell, these are situations when the model falsely equates two states. Such errors are much similar to (Sutton & Barto, 2018, Example 8.3), in which the model erroneously imagine 'bouncing off' of a non-existing wall and thus failing to exploit a shortcut.

From a more general point of view, the problem is another incarnation of the exploration-exploitation dilemma. We highlight this by evaluating our method in sparse reward environments. In a planning context, exploration aims to collect data for improving the model, while exploitation aims to act optimally with respect to the current one. The problem might be solved by encouraging to revisit state-action pairs with an incentive of 'bonus rewards' based on the state-action visitation frequency. Such a solution, (Sutton & Barto, 2018, Dyna-Q+), is restricted to tabular cases. TBV can be though as its scalable version.

Our contributions are:

- TBV - a general risk-aware mechanism of dealing with pessimistic model errors.
- Conceptual sources and types of model errors accompanied with empirical evidence.
- Practical implementation of TBV with: classical Best First Search - a classical graph search algorithm family and Monte Carlo Tree Search - the state-of-the-art class of planners in many challenging domains.
- Empirical verification of TBV behaviour in two sparse rewards domains: ToyMontezumaRevenge Roderick et al. (2018), the Tower of Hanoi puzzle.

The code of our work is available at: `https://github.com/ComradeMisha/TrustButVerify`.

2 RELATED WORK

Many model-based reinforcement learning algorithms follow the framework laid out in Dyna, see Sutton (1991). Kaiser et al. (2020) uses a model to collect fictitious playouts and obtains impressive results for low data regime on Atari. In the continuous domains, a similar approach is adopted by Kurutach et al. (2018). Simultaneous training of a single policy on model ensemble is proposed to tackle model errors. Interestingly, this approach is reported to perform better than back-propagation through time. Clavera et al. (2018) indicates that polices trained on model ensembles might be over-conservative and proposed to use policy meta-learning. Another approach to dealing with model errors works by choosing the unroll horizon for a model. Janner et al. (2019), proposed short model-generated *branched rollouts* starting from data collected on "real environment". Perhaps surprisingly, they find that one-step rollouts provide competitive results. Similarly, short rollouts are used in Feinberg et al. (2018) and Buckman et al. (2018) to improve value estimates. Janner et al. (2019); Buckman et al. (2018) uses ensembles to reduce model bias. The former generates diverse data similarly to Chua et al. (2018), and the latter automatically adapt the planning horizon based on uncertainty estimates. Xiao et al. (2019) proposed another adaptive horizon mechanism based on measurement of model errors using principled Temporal Difference methods.

There is a huge body of work about planning; for classical results, the interested reader is referred to Cormen et al. (2009), Russell & Norvig (2003) and the references therein. Traditional heuristic algorithms such as A^* (Hart et al. (1968)) or GBFS (Doran & Michie (1966)) are widely used in practice. In Agostinelli et al. (2019) the authors utilise the value-function to improve upon the A^* algorithm and solve Rubik's cube. Similarly, Orseau et al. (2018) bases on the classical BFS to build a heuristic search mechanism with theoretical guarantees. The Monte Carlo Tree Search (MCTS) algorithm, which combines heuristic search with learning, led to breakthroughs in the field, see Browne et al. (2012) for an extensive survey. Famously, Silver et al. (2018) develop MCTS-based technique to master the ancient game of go. The POLO algorithm presented in Lowrey et al. (2019) proposes to enrich MPC planning with ensemble-based risk measures to augment exploration. This work is extended by NEEDLE (Milos et al. (2019)) to tree-based MCTS planners and by Lu et al. (2019) to successfully tackle life-long learning scenarios of environmental change. Hamrick et al. (2020) explores how to use better statistics collected while searching to calculate signal for value function training.

The above works plan using a perfect model. Recently, a lot of research has focused on planning with learned models. Nagabandi et al. (2018b) successfully blends the strengths of model-based and model-free approaches. PlaNet (Hafner et al. (2019a)) and Dreamer Hafner et al. (2020) train latent models, which are used for planning. Conceptually, a similar route was explored in MuZero (Schrittwieser et al. (2019)) and Universal Planning Networks (Srinivas et al. (2018)). Farquhar et al. (2017) and Oh et al. (2017) investigate the possibility of creating neural network architectures inspired by the planning algorithms.

When a learned model is unrolled during planning, errors typically accumulate dramatically. Racanière et al. (2017) and Guez et al. (2018) are two approaches to learn the planning mechanism and make it robust to model errors. Sekar et al. (2020) uses ensembles to measure model uncertainty and deal with model inaccuracies. In a somewhat similar spirit, Eysenbach et al. (2019)

treats the replay buffer as a non-parametric model forming a graph and uses ensembles of learned distances as a risk-aware mechanism to avoid certain types of model errors.

There is an emerging body of work on the intersection of model learning and exploration. Pathak et al. (2017) uses a self-supervised inverse dynamics model to provide intrinsic reward driving exploration. Sekar et al. (2020) proposes a method of learning task agnostic global world model, which can be utilised to quickly solve new incoming tasks. Somewhat similarly, Henaff (2019) draws inspiration from E^3 algorithm (Kearns & Singh, 2002), actively searching for data useful in model training. Shyam et al. (2019) has a similar objective implemented using novelty mechanism derived from Bayesian perspective on exploration.

Our work also falls into this area. We construct a model-based approach capable of exploring sparse reward environments. Compared to (Pathak et al., 2017) we use uncertainty measures and planning. Sekar et al. (2020) uses ensembles similarly to us but instead of an on-line planner utilises Dreamer (Hafner et al., 2019b). Henaff (2019) is perhaps the closest to our work, however it puts emphasis on continuous environments and consequently leaves aside using graph structure and related model errors. Similarly, Shyam et al. (2019) largely ignores graph structure and online planning, though briefly mentions a loop problem stemming from this fact.

3 TBV FRAMEWORK

This work attempts to pave the way in harnessing the power of graph search methods to reinforce model-based reinforcement learning. Majority of the planners were designed to solve a well defined problem, which means that the user is expected to supply a *perfect model*. Consequently, when the user fails to do it, the planner can yield an unexpected, or outright wrong, result. We provide an illustration of this phenomenon in Section 3.1, where on example of the Tower of Hanoi, we show that while moderate sizes of the puzzle are easily solved by the planners with perfect model, their performance deteriorates quickly in the presence of model errors. Although in some environments, like games or simulators, a perfect model is available, in most real-world scenarios it is a luxury we cannot afford. Especially in the learning regimes, obtaining the perfect model requires exploration of the whole state space, which is usually unacceptably hard. Consequently, any real-world model-based RL system is unlikely to avoid dealing with model errors.

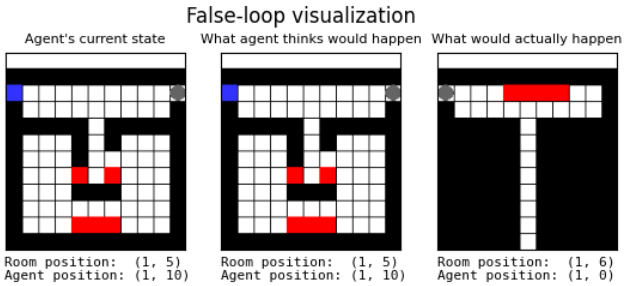

Figure 1: Agent (gray circle) is moving in a grid-world. It can move through the door (blue) to other rooms, if it enter a trap (red) the episode is terminated. in right top corridor). (Genuine) one-step loop arise when agent attempts to walk into the wall. In the example, the model imagines that performing action "right" will not result in environment change (*false loop*). In fact choosing this action would result in moving to the next room.

We are thus led to two seemingly conflicting desiderata for the choice of a method: we would like to use models which are robust to model errors, and planners which leverage the structure of the underlying problem, and hence are more susceptible to model errors. We advocate developing methods, which will facilitate adoption of search algorithms into setting with imperfect models. In this work we make a step towards implementing this plan. We identify and study certain type of model errors, which we dub *false loops*, and we propose *Trust, but verify* (TBV) method, which can successfully immunize planners with respect to said errors.

3.1 MODEL ERRORS

The model errors arise because the agent visits a subset of state-action pairs. This subset grows, which typically improves the situation but causes non-stationarity. This conflicts with supervised learning paradigms that data are i.i.d. Similarly, generalization, which is a great feature of the modern neural network-powered machine learning turns out to be harmful if it occurs prematurely. Sutton &

Barto (2018, Section 8.3) conveniently categorize model errors into: optimistic and pessimistic. A situation when a model predicts higher reward or better state transitions (optimistic error) is easier to handle since while the agent enters an erroneous region, it collects corrective data. More dangerously, predictions more pessimistic then reality may prevent the agent ever to visit certain regions.

The optimistic and pessimistic errors are often of the same nature. In this work, we concentrate on errors related to the graph structure of the problem. By definition, the edges in this graph correspond to transition induced by actions, and thus the errors are "incorrect edges". Perhaps the simplest prediction task (for a model) is to determine if an edge is *one-step loop* (i.e. if an action will change or not the current state). Another task is the prediction of absorbing states, see Figure 1 for examples. In this work, we focused on the loop errors, as they turn out to be critical to performance. In experiments, we also observed other types of errors (like "random teleports"); however they disappeared rather quickly during the training. It is possible that they would be more problematic in other environments, which we leave for further work. In Figure 5 we show numerical results that show rather frequent occurrence of *false loop*. This is a pessimistic error, which, unless corrected, e.g. by TBV, prevent exploration.

Algorithm 1 Model-based training loop

```
# Initialize parameters of ensemble of models
# Initialize parameters planner specific networks
# Initialize buffer
repeat
    episode ← COLLECT_EPISODE
    buffer.ADD(episode)
    B ← buffer.BATCH
    Train ensembles of models
    Update planner specific networks
until convergence
function COLLECT_EPISODE
    s ← env.RESET
    episode ← []
    repeat
        a ← TBV_planner.CHOOSE_ACTION(s)          ▷ TBV-augmented graph planner
        s', r ← env.STEP(a)                       ▷ using models, see Algorithm 2
        episode.APPEND((s, a, r, s'))
        s ← s'
    until episode is done
    return episode
```

3.2 TRUST, BUT VERIFY METHOD

Trust, but verify (TBV) is a method which can be integrated with graph search planners. The key idea of TBV is to prioritize visits in states for which the model is suspected to be pessimistic. In each state the planner is asked for a recommended action. If the model uncertainty of the current state is high, we may suspect that the planner's results are misleading. In such a case, a different action chosen to verify the model's accuracy. This promotes exploration and thus reduces errors in the model. To measure the uncertainty, we utilize ensembles, which is a popular approach in the literature. To avoid potential problems with *a priori* uncontrollable scale of uncertainty, we quantify its magnitude as quantiles of the uncertainty distribution induced by the planner's search graph. The details follow and the pseudo-code for TBV is listed in Algorithm 2. For completeness, we also present a typical model-based training loop in Algorithm 1.

As shown in Algorithm 2, TBV interacts with a planner, which uses an imperfect learned model. During its execution, the planner expands a search graph, g_p, in order to propose an action a_p.Each state-action pair in g_p is given a score reflecting uncertainty assigned to it by the model, and which is quantified by a function DISAGREEMENT_MEASURE. Based on that, TBV decides whether to keep the planner's decision a_p (trust the model) or override it (and explore to verify model predictions). More precisely, a state-action pair with the maximal score is considered if it reaches above the

threshold given by $quantile_score$. We found using quantiles instrumental in avoiding tuning thresholds which are problem-specific and change during the training.

Algorithm 2 TBV planner

Require: $model$ ensemble of models used in Algorithm 1
$planner$ planner that uses $model$
QR quantile rank $\in (0, 1)$
Use: QUANTILE(l, q) computes the q-quantile of list l
RANDOM random number from $\mathcal{U}(0, 1)$

```
function CHOOSE_ACTION(state)
    a_p ← planner.CHOOSE_ACTION(state)
    g_p ← planner.GET_GRAPH_OF_PLANNING
    scores ← []
    for state ∈ g_p do
        scores.APPEND(STATE_SCORE(state))
    quantile_score ← QUANTILE(scores, QR)
    one_step_scores ← []
    for a ∈ actions do
        one_step_scores.APPEND(TRANSITION_SCORE(state, a))
    if max(one_step_scores) > quantile_score and RANDOM > 0.5 then
        return arg max_action one_step_score
    else
        return a_p
function TRANSITION_SCORE(state, action)
    predictions ← []
    for network ∈ model do
        next_state, next_reward ← network.PREDICT_NEXT_STATE(state, action)
        predictions.APPEND((next_state, next_reward))
    return DISAGREEMENT_MEASURE(predictions)
function STATE_SCORE(state)
    return max_action(TRANSITION_SCORE(state, action))
```

Furthermore, QR is relatively easy to tune (see 4) and is the only hyperparameter introduced by TBV. In the algorithm, we use additional randomization to prevent multiple revisits of the same state-action pair, while waiting for the model (and scores) update. It is merely a performance improvement not essential to our method, we leave to further research developing more principled approaches (e.g. confidence intervals).

We note in using TBV yields little computational overhead beyond the necessity of using ensembles (in efficient implementations STATE_SCORE and TRANSITION_SCORE calls can be inlined into $planner$.CHOOSE_ACTION). DISAGREEMENT_MEASURE depends on the state space representation; in our experiments, we found standard deviation computed on sates work well. Other methods, like Bayesian inference, could be used as well.

3.3 BEST FIRST SEARCH

Best First Search (BestFS) is a family of search algorithms that builds a graph by expanding most promising nodes among already visited. A classical BestFS expands nodes until it finds the goal state. In order to use it in on-line planning regime, we choose to extend the graph by fixed amount of vertices in any planing phase (typically 10). The proposed action is the first edge on the shortest path to the best node in the subgraph searched so far.

An important design decision is choosing heuristics for assigning nodes numerical values. This leads to many specific algorithms (e.g., BFS, A*). In our experiments, we concentrated of exploration and used `Disagreement_Measure` (standard deviation) between ensemble predictions. In our experiments, such a version proved to be effective in in discrete problems with spare rewards.

By design BestFS ignores one step loops and thus is especially prone to *false loops* described in Section 1.

3.4 MONTE CARLO TREE SEARCH

Monte Carlo Tree Search (MCTS) is a well-established planning algorithm which was successfully applied to complex problems. Due to its simplicity and effectiveness, it has numerous extensions, see Browne et al. (2012) for a survey, including famous AlphaZero Silver et al. (2018).

A vanilla MCTS constructs a search tree in four stages: node selection, leaf expansion, rollout, and backpropagation (see Browne et al. (2012, Section 3.1) for a comprehensive review). In modern approaches, the rollout phase is often replaced by a neural network evaluation step. MCTS is often equipped with auxiliary mechanisms exploiting the graph structure of the problem. In our work, we use an MCTS implementation with four such mechanisms: hard loop-avoidance inside the search tree, soft loop-avoidance within to the whole episode, the transposition tables and amortized value estimates, see Milos et al. (2019) for details.

Transposition tables (Childs et al., 2008; Gelly et al., 2012) enable sharing information between different tree nodes, which correspond to the same state (i.e. have been reached using distinct trajectories). Similar in spirit are mechanisms, which attempt to avoid visiting a node multiple times. These are akin to classical graph search methods in which a node it typically visited only once. This may take place on the level of the whole episode, within one planning phase (in the search tree). A "soft" way of implementing such mechanisms is to use "virtual loss", which assigns a temporary negative value when a graph node is first encountered. This encourages the exploration and tree building mechanism to avoid this node. Such solutions have been proposed in (Segal, 2010) to enhance parallelism and later used in (McAleer et al., 2019) to tackle Rubik's cube search. A hard version (equivalent to setting virtual loss to $-\infty$) strictly prohibits entering the same node twice. In (Milos et al., 2019) it is reported to be a significant efficiency boost for modest planning budgets on the classical Sokoban puzzle. Recent works (Milos et al., 2019; Hamrick et al., 2020), propose to utilize "amortized value estimates" (which are calculated using the internal MCTS statistics) as targets for value function for training.

Our version uses a learned value function (similarly to AlphaZero) to evaluate new nodes. Similarly to Lowrey et al. (2019) and Milos et al. (2019) we use also value function ensemble to ensure exploration during planning.

4 EXPERIMENTS

The experiments below were chosen to highlight the aforementioned *false-loops* model errors. They demonstrate the effectiveness of TBV. We have selected ToyMontezumaRevenge environment and the Tower of Hanoi puzzle. These environments have sparse rewards and distance to the goal spans at least a few hundred steps. It makes them challenging from the point of view of RL, while at the same time graph algorithms with access to a perfect model can easily brute-force search the state space to find a solution. Since there a gap in performance of the two categories of methods, the environments present a good testing grounds for studying unfavorable influence of model errors on graph planners.

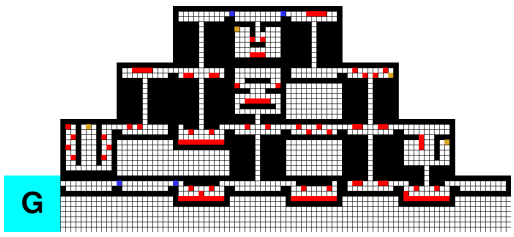

Figure 2: Map of ToyMontezumaRevenge state space. The only source of reward is reaching the goal room marked with G. To obtain it agent needs to gather several keys (marked in yellow) and pass through doors (marked in blue). Stepping on a trap (marked in red) results in episode termination. We use a code from `https://github.com/chrisgrimm/deep_abstract_q_network`

As a performance metrics for the methods, we use the minimal distance to the goal state, measured in steps, and treated as a function of total environment steps used in training. We apply this measure to two planners: BestFS and MCTS (each in the vanilla version, with ε-greedy exploration,

and TBV). For comparison, we also use RND Burda et al. (2018), a strong model-free exploration baseline. In all experiments, we found that planners equipped with TBV mechanism significantly outperform other baselines.

4.1 TOYMONTEZUMAREVENGE

ToyMontezumaRevenge is navigation, maze-like, environment introduced by Roderick et al. (2018) as a testing ground for long-horizon planning and exploration. It has a greatly simplified visual layer compared to the original Montezuma's Revenge Atari game, but it retains much of its exploration difficulty. In our experiments, we consider the biggest map containing 24 rooms and sparse rewards: the agent gets a reward 1 only if it reaches the treasure room, otherwise the episode is terminated after 600 steps. Observation is represented as a tuple containing current room location, agent position within the room, and status of all keys and doors on the board.

For both planners using epsilon-greedy improves over vanilla version but not as much as *TBV*.

It turns out that traps (see caption below Figure 1) pose a challenge for BestFS algorithm. In order to mitigate this issue, we to modified the novelty mechanism to explicitly avoid traps (performing actions resulting in episode termination without positive reward, according to the learned model). For a fair comparison, we also added the same mechanism to epsilon-greedy: when sampling from action space, only actions not leading to traps were considered. Similar improvements were unnecessary for MCTS.

For BestFS, 4 out of 10 experiments were unable to leave the first room. This stemmed from the fact that the agent could not make progress by getting stuck in the early stages of exploration due to the inability to pass through *false-loops*. This is visualized in Figure 3, where it is also shown that TBV helps.

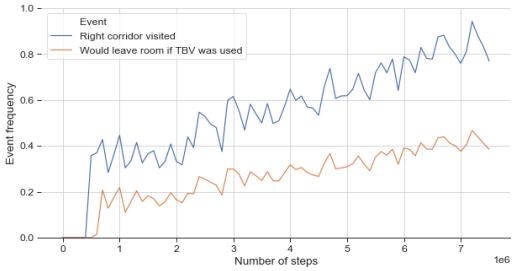

Figure 3: Numerical measurement of influence of TBV on BestFS in the presence of a false loop in ToyMR. The upper (blue) line presents the frequency of reaching the open doors in the first room. The agent without TBV never enters the door due to a false loop error. The lower (orange) line present the frequency the agent would leave the room if TBV was used.

Interestingly, it was quite hard to tune RND on this domain (the agent struggled to find the second key and explored at most 14 out of 24 rooms). The reason for this could be related to the fact that RND needs more data to work, and our low data regime was (originally RND was trained with billions of transitions). Additionally, RND was designed and tested for visual observations, so our setup could not lend to its strengths. Hyperparameters choice for RND choice can be found in Appendix (see Table 3 and Table 4).

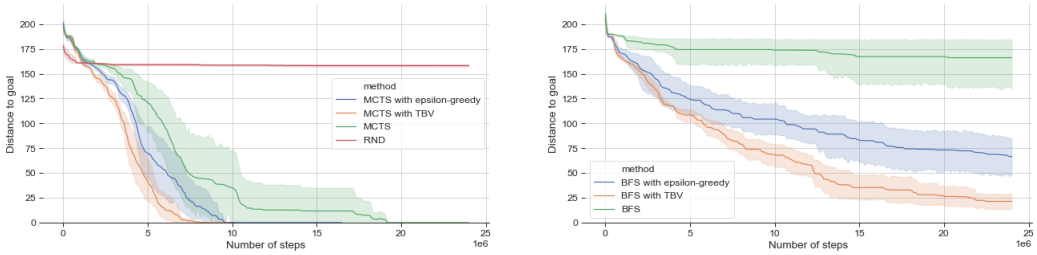

Figure 4: ToyMontezumaRevenge, comparison between planners augmented with *TBV*, epsilon-greedy, and no top-level exploration mechanism.

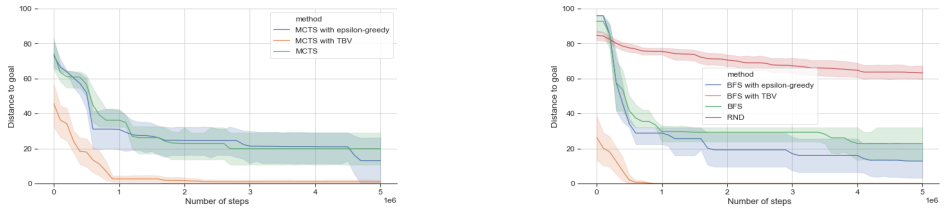

Figure 5: Tower of Hanoi, both MCTS and BestFS quickly find the solution if augmented with *TBV*. In other cases finding solution in 5 million steps is very rare.

4.2 TOWER OF HANOI

The Tower of Hanoi is a classical puzzle consisting of 3 pegs and n disks. The objective of the game is to move the entire stack of disks from the starting peg to the goal peg. The rules are that only one disk can be moved at a time and it is disallowed to put a larger disk on top of a smaller one. This makes it a challenging combinatorial problem, and an optimal solution requires $2^n - 1$ moves. This is an interesting domain for planning methods as model-free algorithms struggle to deal with larger instances (Troussard et al. (2020) used $n = 4$, and Edwards et al. (2018) used $n = 3$). In this paper, we use a considerably harder version, with $n = 7$. Furthermore, we consider a more difficult version of the puzzle, where the agent does not have access to the disk sizes. This does not allow to generalize the dynamics of the environment from partial data, and it makes it challenging to learn the rule that "the smaller disk can always be placed on larger". Instead, the agent needs to learn the disk sizes for each pair of disks separately, as the exploration progresses.

As can be seen in Figure 5, TBV method significantly improves both MCTS and BestFS planners in this domain. This is due to the nature of the Tower of Hanoi, which can create an illusion that the biggest disk stays in the same position. This can result in the false belief of the model that the biggest disk does not move, hence causing the *false-loop* errors.

We found that RND continued to explore the environment when left for longer training (up to 25M transitions). Its progress is, however, very slow when compared to other methods. This is in line with our observation from the previous section that RND is designed to work on larger scale experiments.

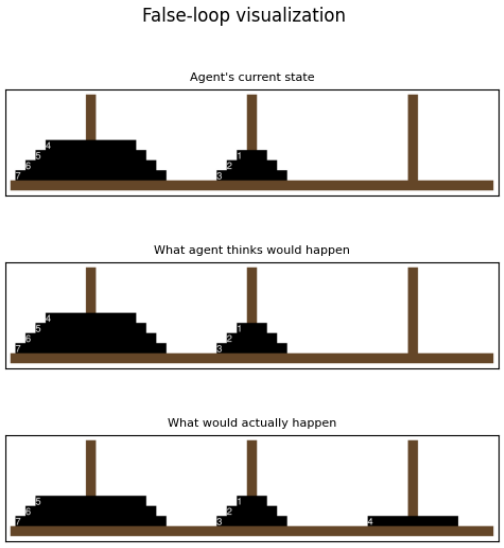

Figure 6: Tower of Hanoi puzzle. For the first time, the agent finds himself in a position to move the fourth disk but mistakenly believes that it is not possible, resulting in a false-loop.

4.3 QUANTILE RANK SENSITIVITY

Quantile Rank (QR) is the only hyperparameter of *TBV*. We found out that values in the range $0.8 - 0.95$ worked across most setups. Excessively large values (e.g. 0.99) make *TBV* a mechanism to change the planner's action rarely. Performance of a BestFS agent with *TBV* across multiple values of QR is presented in Appendix in Figure 7.

REFERENCES

Forest Agostinelli, Stephen McAleer, Alexander Shmakov, and Pierre Baldi. Solving the rubik's cube with deep reinforcement learning and search. Nature Machine Intelligence, 1(8): 356–363, 2019. doi: 10.1038/s42256-019-0070-z. URL https://doi.org/10.1038/s42256-019-0070-z.

Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in games, 4(1):1–43, 2012.

Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In Advances in Neural Information Processing Systems, pp. 8224–8234, 2018.

Yuri Burda, Harrison Edwards, Amos J. Storkey, and Oleg Klimov. Exploration by random network distillation. CoRR, abs/1810.12894, 2018. URL http://arxiv.org/abs/1810.12894.

Benjamin E. Childs, James H. Brodeur, and Levente Kocsis. Transpositions and move groups in monte carlo tree search. In Proceedings of the 2008 IEEE Symposium on Computational Intelligence and Games, CIG 2009, Perth, Australia, 15-18 December, 2008, pp. 389–395, 2008. doi: 10.1109/CIG.2008.5035667. URL https://doi.org/10.1109/CIG.2008.5035667.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In NeurIPS 2018, pp. 4759–4770, 2018.

Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel. Model-based reinforcement learning via meta-policy optimization. In 2nd Annual Conference on Robot Learning, CoRL 2018, Zürich, Switzerland, 29-31 October 2018, Proceedings, volume 87 of Proceedings of Machine Learning Research, pp. 617–629. PMLR, 2018. URL http://proceedings.mlr.press/v87/clavera18a.html.

Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. Introduction to algorithms. MIT press, 2009.

James E Doran and Donald Michie. Experiments with the graph traverser program. Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences, 294(1437):235–259, 1966.

Ashley D Edwards, Laura Downs, and James C Davidson. Forward-backward reinforcement learning. arXiv preprint arXiv:1803.10227, 2018.

Ben Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. Search on the replay buffer: Bridging planning and reinforcement learning. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada, pp. 15220–15231, 2019.

Gregory Farquhar, Tim Rocktäschel, Maximilian Igl, and Shimon Whiteson. Treeqn and atreec: Differentiable tree planning for deep reinforcement learning. CoRR, abs/1710.11417, 2017.

Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I. Jordan, Joseph E. Gonzalez, and Sergey Levine. Model-based value estimation for efficient model-free reinforcement learning. CoRR, abs/1803.00101, 2018. URL http://arxiv.org/abs/1803.00101.

Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. Found. Trends Mach. Learn., 11(3-4):219–354, 2018. doi: 10.1561/2200000071. URL https://doi.org/10.1561/2200000071.

Sylvain Gelly, Levente Kocsis, Marc Schoenauer, Michèle Sebag, David Silver, Csaba Szepesvári, and Olivier Teytaud. The grand challenge of computer go: Monte carlo tree search and extensions. Commun. ACM, 55(3):106–113, 2012. doi: 10.1145/2093548.2093574. URL https://doi.org/10.1145/2093548.2093574.

Arthur Guez, Theophane Weber, Ioannis Antonoglou, Karen Simonyan, Oriol Vinyals, Daan Wierstra, Rémi Munos, and David Silver. Learning to search with MCTSnets. In ICML, 2018.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In International Conference on Machine Learning, pp. 2555–2565. PMLR, 2019a.

Danijar Hafner, Timothy P. Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA, volume 97 of Proceedings of Machine Learning Research, pp. 2555–2565. PMLR, 2019b. URL http://proceedings.mlr.press/v97/hafner19a.html.

Danijar Hafner, Timothy P. Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net, 2020. URL https://openreview.net/forum?id=S1lOTC4tDS.

Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Tobias Pfaff, Theophane Weber, Lars Buesing, and Peter W. Battaglia. Combining q-learning and search with amortized value estimates. In 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net, 2020. URL https://openreview.net/forum?id=SkeAaJrKDS.

Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. IEEE transactions on Systems Science and Cybernetics, 4(2):100–107, 1968.

Mikael Henaff. Explicit explore-exploit algorithms in continuous state spaces. In Advances in Neural Information Processing Systems, pp. 9377–9387, 2019.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In Advances in Neural Information Processing Systems, pp. 12519–12530, 2019.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H. Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model based reinforcement learning for atari. In 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net, 2020. URL https://openreview.net/forum?id=S1xCPJHtDB.

Michael J. Kearns and Satinder P. Singh. Near-optimal reinforcement learning in polynomial time. Mach. Learn., 49(2-3):209–232, 2002. doi: 10.1023/A:1017984413808. URL https://doi.org/10.1023/A:1017984413808.

Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. In 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. OpenReview.net, 2018. URL https://openreview.net/forum?id=SJJinbWRZ.

Kendall Lowrey, Aravind Rajeswaran, Sham M. Kakade, Emanuel Todorov, and Igor Mordatch. Plan online, learn offline: Efficient learning and exploration via model-based control. In 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net, 2019. URL https://openreview.net/forum?id=Byey7n05FQ.

Kevin Lu, Igor Mordatch, and Pieter Abbeel. Adaptive online planning for continual lifelong learning. CoRR, abs/1912.01188, 2019. URL http://arxiv.org/abs/1912.01188.

Stephen McAleer, Forest Agostinelli, Alexander Shmakov, and Pierre Baldi. Solving the rubik's cube with approximate policy iteration. In 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019, 2019. URL https://openreview.net/forum?id=Hyfn2jCcKm.

Piotr Milos, Lukasz Kucinski, Konrad Czechowski, Piotr Kozakowski, and Maciej Klimek. Uncertainty-sensitive learning and planning with ensembles. CoRR, abs/1912.09996, 2019. URL http://arxiv.org/abs/1912.09996.

Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In 2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018, pp. 7559–7566. IEEE, 2018a. doi: 10.1109/ICRA.2018.8463189. URL https://doi.org/10.1109/ICRA.2018.8463189.

Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In 2018 IEEE International Conference on Robotics and Automation (ICRA), pp. 7559–7566. IEEE, 2018b.

Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA, pp. 6118–6128, 2017. URL http://papers.nips.cc/paper/7192-value-prediction-network.

Laurent Orseau, Levi Lelis, Tor Lattimore, and Theophane Weber. Single-agent policy tree search with guarantees. In Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada., pp. 3205–3215, 2018. URL https://papers.nips.cc/paper/7582-single-agent-policy-tree-search-with-guarantees.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In Doina Precup and Yee Whye Teh (eds.), Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017, volume 70 of Proceedings of Machine Learning Research, pp. 2778–2787. PMLR, 2017. URL http://proceedings.mlr.press/v70/pathak17a.html.

Sébastien Racanière, Theophane Weber, David P. Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, Razvan Pascanu, Peter Battaglia, Demis Hassabis, David Silver, and Daan Wierstra. Imagination-augmented agents for deep reinforcement learning. In NIPS, 2017.

Melrose Roderick, Christopher Grimm, and Stefanie Tellex. Deep abstract q-networks. In Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018, pp. 131–138, 2018. URL http://dl.acm.org/citation.cfm?id=3237409.

Stuart J. Russell and Peter Norvig. Artificial intelligence - a modern approach, 2nd Edition. Prentice Hall series in artificial intelligence. Prentice Hall, 2003. ISBN 0130803022. URL https://www.worldcat.org/oclc/314283679.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. CoRR, abs/1911.08265, 2019. URL http://arxiv.org/abs/1911.08265.

Richard B. Segal. On the scalability of parallel UCT. In Computers and Games - 7th International Conference, CG 2010, Kanazawa, Japan, September 24-26, 2010, Revised Selected Papers, pp. 36–47, 2010. doi: 10.1007/978-3-642-17928-0_4. URL https://doi.org/10.1007/978-3-642-17928-0_4.

Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. CoRR, abs/2005.05960, 2020. URL https://arxiv.org/abs/2005.05960.

Pranav Shyam, Wojciech Jaśkowski, and Faustino Gomez. Model-based active exploration. In International Conference on Machine Learning, pp. 5779–5788, 2019.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science, 1144:1140–1144, 2018.

Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks. CoRR, abs/1804.00645, 2018. URL http://arxiv.org/abs/1804.00645.

Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. ACM Sigart Bulletin, 2(4):160–163, 1991.

Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. MIT press, 2018.

Martin Troussard, Emmanuel Pignat, Parameswaran Kamalaruban, Sylvain Calinon, and Volkan Cevher. Interaction-limited inverse reinforcement learning. arXiv preprint arXiv:2007.00425, 2020.

Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking model-based reinforcement learning. CoRR, abs/1907.02057, 2019. URL http://arxiv.org/abs/1907.02057.

Chenjun Xiao, Yifan Wu, Chen Ma, Dale Schuurmans, and Martin Müller. Learning to combat compounding-error in model-based reinforcement learning. CoRR, abs/1912.11206, 2019. URL http://arxiv.org/abs/1912.11206.

A APPENDIX

A.1 TRAINING SETUP

Code for all our experiments can be accessed at `https://github.com/ComradeMisha/TrustButVerify`.

Our experiments adhere to the general model-based training loop logic, described in Algorithm 1. We use a distributed system with 32 workers solving distinct episodes, where data gathered across a batch of workers is collected in two common experience buffers: replay buffer for trainable model of size 50000 and replay buffer for value function of size 30000. Before the actual episode solving we collect 1000 random trajectories, that we use for initial training of the model of environment. This initial training has critical importance for performance of our agents. Episodes are limited to: 600 steps for ToyMR and 1000 steps for the Tower of Hanoi.

A.2 NETWORK ARCHITECTURES FOR MODEL AND VALUE

A.2.1 SINGLE NETWORKS

In every experiments we use two neural network architectures: one to estimate the value function of a given state and the other to predict the outcome of taking a step in the environment. In ToyMR we represent state as a vector of length 17 and in the Tower of Hanoi as a vector of length $3 \times$ number of discs (it is due to one-hot encoding of the peg number for each disc).

In both environments, for value estimation we use an MLP architecture with two hidden layers of 50 neurons and ReLu non-linearity and for model we use an MLP with four hidden layers of 250 neurons and ReLU activations.

To update value network for MCTS we follow approach of (Milos et al., 2019) (similar to Hamrick et al. (2020)) for BFS experiments we simply calculate future discounted returns for each transition encountered by agent.

The environment model output consists of: the difference between next and current observations (delta target), reward and a and episode termination flag (0 or 1).

A.2.2 ENSEMBLE

For both value estimation and model we use ensemble of networks with architectures described in Section A.2.1. The final estimation of value is an average of predictions across ensemble. The standard deviation of this predictions multiplied by κ is used by MCTS as an auxiliary score added to value.

In Algorithm 3 we present the procedure of transforming the output of ensemble of model networks to a valid prediction of environment signal. In our experiments, it turned out to be beneficial to train the model to predict change between observations rather then the ready observations. The transformation performs the following step: collects predictions of all networks in ensemble, then averages next predicted change of observations, predicted rewards and predicted end of episode flags. Averaged observation change is added to current state, then clipped to the range of possible values (some coordinates of the state vector are binary variables), rounded to integers and returned as the next predicted state. Reward and end of episode flag are predicted to be true if their average value across networks in ensemble is larger than 0.5.

Algorithm 3 Transforming model ensemble predictions

```
function PREDICT_STEP(state, action)
    next_observations ← []
    rewards ← []
    is_done_flags ← []
    for network ∈ ensemble do:
        (next_obs_delta, reward, is_done) ← network.PREDICT(state, action)
        next_obs ← state + next_obs_delta
        next_observations.APPEND(next_obs)
        rewards.APPEND(reward)
        is_done_flags.APPEND(is_done)
    predicted_reward ← AVERAGE(next_reward)
    if predicted_reward > 0.5 then
        predicted_reward ← 1
    else
        predicted_reward ← 0
    predicted_is_done ← AVERAGE(is_done_flags)
    if predicted_is_done > 0.5 then
        predicted_is_done ← True
    else
        predicted_is_done ← False
    predicted_next_obs ← AVERAGE(next_observations)
    predicted_next_obs ← CLIP(predicted_next_obs)
    predicted_next_obs ← ROUND(predicted_next_obs)
    return(predicted_next_obs, predicted_reward predicted_is_done)
```

To stabilize performance of value and model ensembles we used additional masks mechanism: we create a number of networks (given by ensemble size parameter), but each worker uses only a random subset of given size (given by number of masks parameter).

A.3 Quantile Rank value analysis

Figure 7 presents performance of agent for different values of Quantile Rank.

A.4 Hyper-parameters

In table 3 we present hyper-parameters used in our experiments.

For MCTS we took hyperparameters from Milos et al. (2019). As they used setup without learned model we needed to tune model architecture and model training parameters on ToyMontezumaRevenge domain. Then, we separately tuned epsilon and Quantile Rank for each combination of domain/planner.

A.5 RND and PPO

RND continued to further explore state space for Hanoi (figure 8), for comparison we also show performance of PPO (the same parameters as RND but without intrinsic reward).

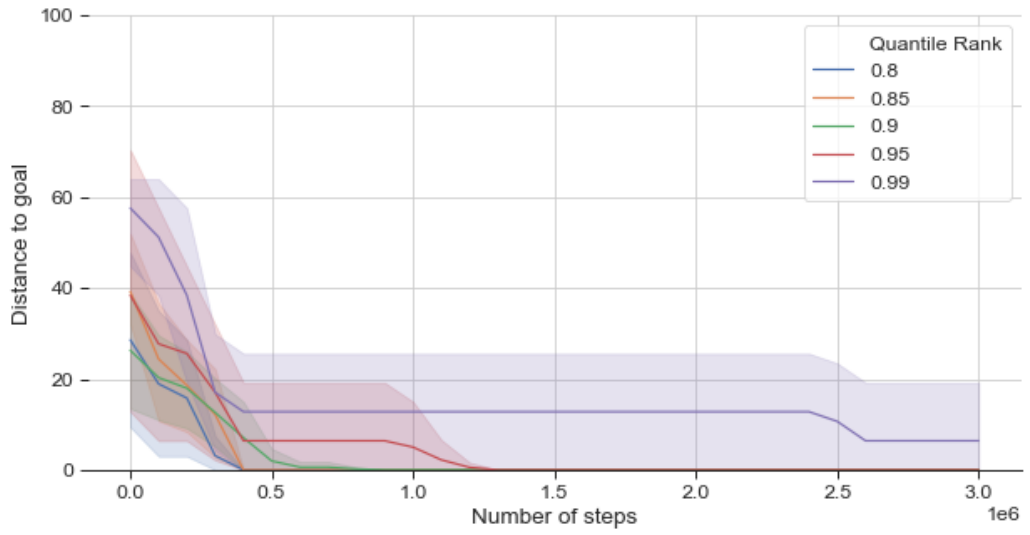

Figure 7: Performance of *TBV* BestFS on the Tower of Hanoi domain with different values of Qantile Rank

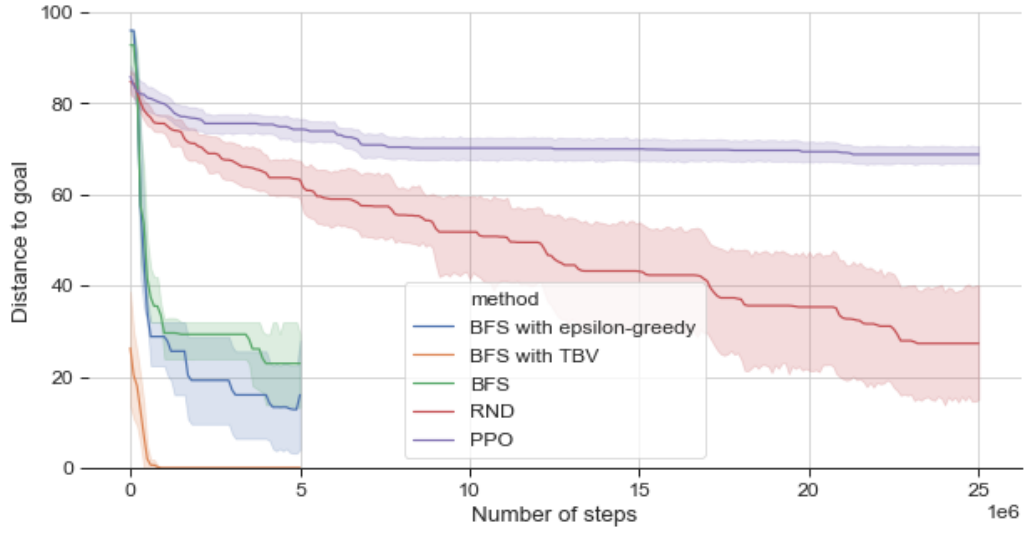

Figure 8: Performance of RND and PPO on Tower of Hanoi with longer training.

Parameter	Toy MR		Tower of Hanoi	
	BestFS	MCTS	BestFS	MCTS
Number of planner passes C	10	10	10	10
Discounting factor γ	0.99	0.99	0.99	0.99
ε greedy exploration for baslines	0.001	0.02	0.001	0.02
score factor for value uncertainty κ	-	3	-	3
Dead end value	-	-0.2	-	-0.2
Avoid history coefficient [1]	-100	-0.2	-100	-0.2
Value ensemble size [2]	20	20	20	20
Value ensemble mask size [2]	10	10	10	10
Model ensemble size [2]	8	8	8	8
Model ensemble mask size [2]	4	4	4	4
Optimizer	RMSprop	RMSprop	RMSprop	RMSprop
Learning rate	2.5e-4	2.5e-4	2.5e-4	2.5e-4
Batch size for value training	32	32	32	32
Batch size for model training	1024	1024	1024	1024

[1] The difference between the value of this parameter for BestFS and MCTS has no practical reason, it is an artifact of some tests, however we verified that the exact value of it is not important as long as the value negative.
[2] See Section A.2.2 for explanation of these parameters.

Table 1: Hyper-parameters values used in our experiments.

Parameter	Value
Rollout length	128
Maximal length of an episode	1000
Total number of rollouts per environment	6200
Number of minibatches	4
Number of optimization epochs	4
Coefficient of extrinsic reward	1
Coefficient of intrinsic reward	10
Number of parallel environments	32
Learning rate	0.001
Optimization algorithm	Adam
λ	0.95
Entropy coefficient	0.001
Proportion of experience used for training predictor	1.0
γ_E	0.999
γ_I	0.99
Clip range	[0.9, 1.1]
Policy architecture	FCN

Table 2: Default hyper-parameters for PPO and RND algorithms for Hanoi experiments where applicable.

Parameter	Value
Rollout length	128
Maximal length of an episode	600
Total number of rollouts per environment	6200
Number of minibatches	4
Number of optimization epochs	16
Coefficient of extrinsic reward	1
Coefficient of intrinsic reward	100
Number of parallel environments	32
Learning rate	0.001
Optimization algorithm	Adam
λ	0.95
Entropy coefficient	0.001
Proportion of experience used for training predictor	1.0
γ_E	0.999
γ_I	0.99
Clip range	[0.9, 1.1]
Policy architecture	FCN

Table 3: Default hyper-parameters for PPO and RND algorithms for ToyMontezumaRevenge experiments where applicable.

Parameter	Value
Number of optimization epochs	[1, 4, 16]
Coefficient of intrinsic reward	[1, 3, 10, 30, 100, 300]
Learning rate	$[5 \cdot 10^{-2}, 10^{-2}, 5 \cdot 10^{-3}, 10^{-3}, 5 \cdot 10^{-4}, 10^{-4}]$
λ	[0.95, 0.99]
Proportion of experience used for training predictor	[0.25, 1.0]
γ_E	[0.999, 0.9999]
γ_I	[0.99, 0.999]
Policy architecture layers number	[2, 3, 4]
Policy architecture layers width	[64, 128, 256]
Random target and prediction networks last layer width	[64, 128]

Table 4: Hyper-parameters for RND algorithm checked during tuning process of ToyMontezumaRevenge.