Exploration When Everything Looks New

Effect of the Local Uncertainty Source on Exploration

Viliam Vadocz[1]
Supervisors: Matthijs Spaan[1]**, Yaniv Oren**[1]
[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Name of the student: Viliam Vadocz
Final project course: CSE3000 Research Project
Thesis committee: Matthijs Spaan, Yaniv Oren, Neil Yorke-Smith

Abstract

Agents improve by interacting with an environment and planning. By leveraging information about what they don't know, they can learn better and faster, at least in environments that benefit from exploring. They do this by estimating the uncertainty in their predictions. There are choices for how to estimate the uncertainty, and in this work, we look at what effect this choice has on the exploration and strength of agents playing board games. We compare the effect of a source of uncertainty which perfectly tracks what the agent has seen, and a source which generalizes. We also describe the challenges associated with tuning uncertainty estimators and show what considerations have to be made when exploration is not all you need.

1 Introduction

We require autonomous agents. One way to create such agents is through reinforcement learning (RL) [42], in which agents improve by interaction with the environment and by receiving rewards for doing well. Agents need to explore the environment to find out which actions give them reward. In the AlphaZero family of RL algorithms, agents plan during interaction to take more informed actions, as well as to improve their understanding of the state and the environment as a whole [39][37][12]. This improved understanding is used to improve the agent so that future interactions and planning are even better. Agents produced by this process are some of the best, especially in the domain of board games. Games are used to test RL algorithms since they provide difficult, yet controlled challenges.

AlphaZero uses a planning algorithm called Monte Carlo Tree Search (MCTS) [8][14]. MCTS considers future reachable states and propagates information back to the current state. In the case of AlphaZero, the information MCTS propagates comes from neural networks. These networks take in a state and output how "good" the state is (called the value of the state) and a policy over the possible actions. The policy associates each action with a value (a probability of taking the action) saying how good that action is in the particular state according to the agent. Both the policy and value are used during planning. The policy dictates which actions are interesting and should be looked at first, and the value is used to determine whether the reached state is good. By planning, both the value and the policy at the current state are improved [15].

The value and policy predictions come from neural networks which can give unpredictable outputs for states which have not been trained on. This means that there is an associated uncertainty with the network outputs, which is low for states that were trained on, and high for states which are unknown. This type of uncertainty is called epistemic, since it comes from a lack of knowledge about a given state [18]. Quantifying epistemic uncertainty remains an open problem because it requires identifying how neural networks generalize, although approaches exist for estimating the uncertainty [32][28][27].

When we plan with uncertain predictions, the conclusions will also be uncertain. MCTS does not propagate the uncertainty from network outputs. This is a missed opportunity because the uncertainty can inform the agent's decisions in the environment: For example, it may be helpful to take actions with higher uncertainty since they contain novel information to learn from [25]. Epistemic Monte Carlo Tree Search (E-MCTS) [29] modifies MCTS to propagate uncertainty from future predictions back to the current state. The consequence is that the agent can associate a confidence in the value of each action, based on the uncertainty from predictions for possible future states.

It is necessary to perform some sort of exploration so that the agent can learn new things.

AlphaZero approaches exploration by adding random noise to the policy at every step, as well as by picking a random action during interaction. The random action is picked proportional to how good the agent thinks the action is (which is reflected in the visitations to each action during planning). Both of these approaches are examples of undirected exploration because they try to randomly stumble into new and interesting states. This is contrasted by directed exploration which actively uses information relevant to exploration, such as the uncertainty.

Directed exploration has been shown to induce learning when undirected exploration fails to do the same [31][30][43][33][29], especially in hard exploration-focused environments, such as mazes. These environments are characterized by having a single hard-to-reach reward. The type of exploration which benefits RL agents in more complex environments, such as board games, is less understood. With E-MCTS, AlphaZero-like algorithms can propagate uncertainty during planning to then choose actions with a bias towards higher uncertainty, following the Mantra of "optimism in the face of uncertainty" [24]. Exploration with E-MCTS has worked remarkably in typical hard-exploration environments, but remains untested in two-player zero-sum board games, for which AlphaZero is known. Exploration in these environments could be beneficial to find strong strategies more quickly or to make the agent resilient in a larger set of the state space [46].

Games like Chess and Go are interesting environments because the adversary has as much power to decide the future states as the agent does, and so it is important for the agent to be robust in a large variety of situations. These environments also require long-term planning, as the outcome of actions can often be seen only much later. Both properties are seen in real-world problems, which makes them relevant to try to tackle. In board games there is not just one desired goal state, but many. These environments have a practically infinite state space which is infeasible to explore fully, and exploration can be punished by the opponent, so we need to consider the trade-off between doing what the agent believes is best (exploitation) and trying new things (exploration) carefully.

One design choice which has an impact on the trade-off and the kind of exploration that is performed is the source of local uncertainty. Local uncertainty refers to the epistemic uncertainty estimate for an agent at a given state. If we use uncertainty driven exploration, the source of uncertainty determines what is considered interesting to explore. A source of uncertainty which generalizes over some feature of the environment means that once that feature is examined, it ceases to be a source of uncertainty, and no longer drives exploration. What kind of local uncertainty is beneficial in the domain of two-player zero-sum games is unknown.

We examine the effect of directed exploration with E-MCTS in a two-player, zero-sum, deterministic, discrete environment, *Tak* [44], as well as the effect of the source of local uncertainty. Specifically, we consider one aspect in which sources of local novelty differ, which is whether they try to be truly epistemic, tracking exactly the states which were seen, or whether they attempt to generalize in some way. We do this by training agents that use hash-based uncertainty estimators: one powered by a conventional hash which tries to avoid hash collisions, and a locality-sensitive hashing method which gives similar hashes to similar states. The latter is an example of a generalizing source of uncertainty. Hashes were used since they are easy to evaluate and tune compared to neural network based uncertainty sources such as random network distillation (RND) [6] used in prior work.

The contributions presented in this papers are as follows:

1. We describe the challenges and pitfalls associated with choosing and tuning uncertainty estimators;

2. we justify the design of the experimental setup motivating our choices for directed exploration with AlphaZero in board games;

3. we examine the effect of directed exploration with E-MCTS in AlphaZero for a two-player zero-sum discrete environment;

4. and finally, we cover the effect of using a source of local uncertainty that is truly epistemic compared to one that generalizes.

To examine the effect of directed exploration we look at the playing strength of the agents during training and the ratio of unique positions encountered throughout training. We empirically show how the two different sources of local uncertainty generalize by seeing how much uncertainty is estimated for unseen states drawn from different distributions. To showcase the difficulty of tuning uncertainty estimators we describe how even after significant effort to tune RND, it fails to discern between seen and unseen states, and has undesirable behaviour with respect to the uncertainty it predicts for different kind of states.

2 Background

We give a formal definition of reinforcement learning including relevant notation, followed by a description of the algorithms and techniques which we used, such as E-MCTS [29], AlphaZero [39], Reanalyze [38], Sequential Halving [20], and RND [6]. We also cover the two hashing algorithms, SimHash [7] and an linear congruential generator (LCG) [21] based hash, which form the sources of local uncertainty for our experiments.

2.1 Reinforcement Learning

Reinforcement learning [42] consists of an agent interacting with an environment, trying to maximize the cumulative reward it receives. The environment is modelled as a Markov Decision Process (MDP) [3]. At any point, the agent is in a state s, and can take an action a out of the possible actions in the state $A(s)$. After taking an action, the agent transitions to the new state s' and receives a reward r. Transitions and rewards may be stochastic, so we annotate the probability of transitioning to s' with reward r when taking action a in state s as $p(s', r|s, a)$. This probability function defines the dynamics of the environment.

The agent interacts with the environment in episodes, which consist of discrete time steps t. The state, action, and reward at time t are written as S_t, A_t, R_t, respectively. The episode ends at time T with the terminal state S_T. The agent acts in the environment according to a policy π. The policy can be deterministic, in which case $A_t = \pi(S_t)$, or stochastic, where $\pi(a|s)$ is the probability of taking action a in state s.

A sequence of states and actions in an episode is called a trajectory, represented as $(S_t, A_t, S_{t+1}, S_{t+2}, ..., S_T)$. The return G_t for a trajectory is defined as the discounted sum of rewards, i.e. $G_t = \sum_{i=0}^{T-t} \gamma^i R_{t+i} = R_t + \gamma G_{t+1}$, where $\gamma \in [0, 1]$ is the discount rate used to devalue future rewards. The policy, transitions, and rewards may be stochastic, which is why we define the value for a policy π and state s as the expected return: $v_\pi(s) = E[G_t|S_t = s]$. Similarly, we define the action-value $q_\pi(s, a) = E[G_t|S_t = s, A_t = a]$. The goal of reinforcement learning is to find the optimal policy which maximizes the value. The value of the optimal policy is written as $v_*(s)$.

Our experiments use *Tak* as the test environment. *Tak* is a deterministic, discrete-action, two-player, zero-sum abstract strategy game like Chess and Go. Deterministic means that

the current state and action determine the next state and reward exactly. More formally, the transition probability $p(s', r|s, a)$ is equal to 1 for exactly one pair of next state s' and reward r, and 0 for all other pairs, for any given state s and action a. Discrete-action means that the state space $A(s)$ if finite for all s. Two-player zero-sum refers to the fact that two agents alternate taking actions in the state, and that the benefit of one agent is an equal deficit for the other. From the perspective of one agent, it is possible to model the adversary as part of the environment, so that the transitions are once again stochastic, and $p(s', r|s, a)$ depends on the adversary. In essence, every opponent defines different environment dynamics. Finally, in abstract strategy board games, there is no reward for actions during episodes except for the final action which leads to a terminal state. Wins receive a reward of 1, losses -1, and draws 0.

2.2 MCTS, Bandits, and E-MCTS

Monte Carlo Tree Search (MCTS) [8] is a planning algorithm which tries to approximate the value of a state $v_*(s)$ by simulating possible trajectories from that state. MCTS iteratively constructs a tree, where the nodes are states in the environment, and actions form the edges from one state to the resulting state after taking said action[1]. The root node is the current state. Every iteration of MCTS consists of three main steps: recursively *select* actions until a leaf node is reached, *expand* the tree by adding a new leaf node along with a new value approximation, and *backup* the value up the tree.

The earliest variations of MCTS approximate the value by simulating random action sequences all the way to terminal states [4][14]. More recent versions of MCTS truncate the simulations at the newly added leaf node and use a heuristic. The heuristic may come from a neural network, in which case we label it $v_\theta(s)$ for some network parameters θ. The value is then propagated back up the tree, adjusting the values $v_{mcts}(s)$ stored at each node along the path. The backup strategy is likewise configurable [9][19], but the standard is to use the mean of propagated values $v_{mcts}(s) = \frac{\sum_{a \in \mathcal{A}(s)} N(s,a) q_{mcts}(s,a)}{N(s)}$ (which can be likened to the Bellman equation), where $N(s, a)$ counts the number of times the action has been visited.

Action selection is modelled as a multi-armed bandit problem [40], where during each iteration of MCTS we want to balance between picking promising actions with high value estimates (simulating good trajectories) and finding new interesting actions. Which actions are good is not known ahead of time, so multiple must be tried before committing. Bandit algorithms attempt to minimize regret, which is the lost reward when compared to choosing the optimal action every time. Upper Confidence bound for Trees (UCT) [22] is a popular algorithm that selects actions based on whichever maximizes the value of a formula of the action-values $q(s, a)$ and visit count $N(s, a)$. PUCT (Predictor + UCT) [36] modifies the UCT formula with prior information about which actions might be good. This policy prior may come from a neural network, which we would annotate as π_θ.

Epistemic Monte Carlo Tree Search (E-MCTS) [29] modifies MCTS by allowing it to estimate and propagate epistemic uncertainty. When a neural network is used to predict the value of a state, there is a distribution of $v_\theta(s)$ which are consistent with what the network has been trained on. The variance of this distribution $\sigma^2(s)$ is how we interpret epistemic

[1]This definition could result in an arbitrary graph depending on the environment dynamics, so we additionally impose that different action sequences which reach the same state in the environment are different nodes in the tree. A variation of MCTS called Monte Carlo Graph Search (MCGS) [11] can work with directed acyclic graphs (DAGs) which occur in the case of transpositions (reaching the same state by two different action sequences).

uncertainty. We estimate the uncertainty using two sources: the local uncertainty and the Uncertainty Bellman Equation (UBE) [34]. UBE approximates an upper bound on the total uncertainty along the future trajectory of the policy, i.e. the uncertainty which will be reached in the future when following the policy.

The combination of these two sources lets us categorize states. When the local uncertainty is high, the state is new. When the local uncertainty is low but UBE is high, it means that the current state is not new, but new states are reachable from it. When both the local uncertainty and UBE is low then the state is a "dead end" in terms of novelty. UBE is approximated by a neural network, written as u_θ, and so it is also susceptible to epistemic errors. That is why low UBE should be ignored in the case of high local uncertainty. We combine the two sources by taking the maximum: $\sigma^2(s) = max(\texttt{local\_uncertainty}(s), u_\theta(s))^2$. The propagated uncertainty is accumulated at each node as the mean standard deviation $\sigma_{mcts}(s)$.

The propagated uncertainty can be used for directing exploration. E-UCT is a modification of UCT [22] which replaces the the action-value $q(s, a)$ in the formula by the optimistic action value $q(s, a) + \beta\sigma_{mcts}(s)$, where $\beta \geq 0$ is an exploration hyper-parameter. By using a predictor we get E-PUCT. The result of adding the standard deviation is that actions with higher uncertainty will be visited more.

2.3 AlphaZero and Reanalyze

Consider an agent which produces state value approximations $v_\theta(s)$ and a prior policy $\pi_\theta(s)$ for any state s using a neural network. By planning with MCTS, it produces an approximation $v_{mcts}(s)$ of the value for a state, approximate action-values $q_{mcts}(s, a)$ for each action, as well as the visits to each action $N(s, a)$. Because visits were allocated to each action according to a bandit algorithm, they correlate with how good each action is[3], and the visit distribution can be used as a policy for acting in the environment. In fact, MCTS with PUCT acts as a policy improvement operator that takes a prior policy $\pi_\theta(s)$ and makes it better [15]. We can use the improved policy to update the network parameters θ to predict better policies in the future. This is the core idea behind AlphaZero.

AlphaZero (AZ) [39] and its follow-up algorithms such as MuZero (MZ) [37] and Gumbel MuZero (GMZ) [12] are able to learn without prior knowledge about what is good. A bad agent can act as a slightly better agent with search, which can be distilled into a neural network so we end up with that slightly better agent with which to repeat the process. This search-guided interaction in the environment is referred to as self-play. The result of self-play is targets, which include the improved policy and the value target. In AZ the improved policy is recovered from the visit counts $N(s, a)$ to each action from the root, and the value target comes from the result of the game, or more formally the return G_t, which is just the discounted terminal reward $\gamma^{T-t}R_T$ in the case of typical abstract strategy game environments.

Sample efficiency refers to how strong of a policy we are able to create per amount of interactions in the environment. Sample efficiency is improved by using reanalyze [38], which is a process which is able to take old targets and renew them. This is done by sampling an old target and by redoing the planning at that state using the newest agent, which produces an updated improved policy. The value target is different than in self-play, since

[2]We also clip the size between 0 and the maximum variance, which is 4 in the case of a value that ranges from -1 and 1.

[3]At least in the case of bandit algorithms which minimize cumulative regret. For algorithms which minimize simple regret this is not true. This becomes relevant when we discuss Sequential Halving [20].

this is done outside of an episode and there is no terminal reward. Several choices exist for how to get the value target, one of which is temporal difference (TD) learning [41]. TD learning means that the target for the current state is the discounted prediction for the state that comes later in the episode (n steps for n-step TD learning). The future states used for TD-learning in reanalyze are those taken from earlier self-play episodes. In notation: $v_{target}(S_t) = \gamma^n v_\theta(S_{t+n})$.

The states reached by earlier policies might be quite different than those the current policy would reach. This essentially means that reanalyze is a way to learn from off-policy data, without suffering from the common pitfalls associated with off-policy learning [26]. Reviewing older episodes might also help address catastrophic forgetting [13]. When exploring according to an uncertainty-seeking policy produced by E-MCTS + E-PUCT, the agent generates off-policy data. Reanalyze provides a way to learn from exploratory episodes.

2.4 Gumbel MuZero and Sequential Halving

Gumbel MuZero [12] is named such for its use of the Gumbel top-k trick [23]. The trick allows sampling k actions from the policy distribution without replacement in an efficient way by sampling Gumbel noise for each action. We sample actions to concentrate planning on fewer actions so that they are searched deeply even with low planning budgets [17]. The actions which are left out do not get analyzed at all, and if we were to use the improved policy based on visit counts, these actions would receive a probability of 0. This is undesirable because we technically have no information about those actions, which is why GMZ introduces an improved policy formula based on the action-values $q_{mcts}(s, a)$, using the root value prediction $v_\theta(s)$ as an approximation for the unvisited actions.

GMZ made another interesting choice regarding the selection algorithm at the root of search. MCTS typically uses bandit algorithms which try to minimize cumulative regret, but when selecting an action to take in the environment we do not care about the rewards reached during search. We only care about the reward of the picked action. This is the minimization of simple regret, and referred to as "pure exploration" [5]. GMZ chooses to use Sequential Halving [20]. Sequential Halving is an aptly named algorithm because it sequentially halves the actions that it considers until it arrives at one action that it deems best. At each step of the halving it visits all actions equally often, and then removes the worse half based on the prior policy π_θ and the action-values $q_{mcts}(s, a)$. When using E-MCTS with Sequential Halving it is possible to select for actions which maximize for the optimistic action-value $q_{mcts}(s, a) + \beta\sigma_{mcts}(s)$.

2.5 RND, SimHash, and LCGs

Random Network Distillation (RND) [6] is a technique that can be used for local uncertainty quantification. It uses two randomly initialized networks, a *target* and a *predictor*. As part of training, the predictor's weights are adjusted to minimize the mean squared error (MSE) between the outputs of the two networks. The target network is not modified. The idea is that for states which are unobserved, the difference between the outputs of the two networks is large, signifying high epistemic uncertainty, while for states which have been trained on, the MSE is low.

RND is a relatively cheap (computationally-speaking) source of uncertainty, which can scale to handle arbitrarily large state spaces. Unfortunately, it has a considerable downside in being very difficult to tune. The predictor generalizes to some degree, which means the

MSE between its outputs and the outputs of the *target* will tend to decrease during training, even for unseen states. This means some form of normalization is required on the output to maintain a difference between seen and unseen states.

SimHash [7] is a locality sensitive hashing technique. It has the property that the probability that a specific bit in the hashes of two states is the same is proportional to the angle between the two state vectors. This means that states which have a similar state vector direction will have similar hashes. When used as a source of epistemic uncertainty, this produces an arbitrary generalization which does not require any domain knowledge, but it is affected much by the state vector representation. SimHash has been successfully used for exploration in hash-explore [43]. Hashes extend simple count-based and pseudo-count exploration [2] [33] to large state-spaces for which it is infeasible to keep a count for each possible state in memory.

Linear[4] congruential generators (LCG) [21] are a family of recurrence relations of the form $x_{n+1} = (ax_n + c) \pmod m$, $i \geq 0$, $m > x_n \geq 0$. The sequences produced by these recurrence relations are used for pseudo-random number generation [35]. The recurrence formula always produces the same output for the same input, and the sequence produces values which are uniformly distributed, which makes this recurrence relation also useful for simple conventional, albeit insecure, hashing. Since it is a conventional hashing technique, it produces different hashes even for similar inputs, and so it can be used as an almost perfect source of epistemic uncertainty (imperfections come from unlikely hash collisions).

3 Method

3.1 Tuning Sources of Local Uncertainty

A source of local uncertainty is required to explore with E-MCTS. RND [6] has been used successfully in prior work, although it must be tuned for best results. The process of tuning RND consisted of three main challenges: output normalization, input normalization, and distinction between "seen" (states that were trained on) and "unseen" states (states that were not trained on).

We tried normalizing the output using a minimum and maximum like $RND_{normalized} = \frac{RND_{out} - RND_{min}}{RND_{min} - RND_{max}}$, where the minimum and maximum come from outputs seen during planning, or from a reference batch of states. Although it seems sensible, this is actually a bad way to normalize, because the smallest and largest inputs come from outliers which are far from other reasonable outputs. The result of this normalization is that it "squishes" all other outputs into a small range which is undesirable. Dividing by the running standard deviation works better.

The RND output is sensitive to the input size. Larger input size tends to give a larger output, regardless of whether the state has been seen. The standard approach of subtracting the element-wise running mean from the input and then dividing by the running standard deviation does not work well for board game inputs. The input representation is typically a boolean tensor with many zeroes which might change to ones only rarely. Diving by the running standard deviation for such an element will only amplify the problem where larger inputs give a larger output.

To address the issue of input size effecting output size, we tried several normalization approaches: resize the input so that the magnitude is equal to one, resize per layer, apply a

[4]The convention is to call them linear even though when $c \neq 0$ the recurrence relation is an affine transformation.

constant random noise to the inputs, modify the network architecture to include a *LayerNorm* [1] at the start, and more. None were particularly effective at solving the problem.

For RND to work as an uncertainty estimator at all, it is crucial that it distinguishes seen and unseen states. We observed that outputs for seen and unseen states decreased at about the same pace, and that the difference between them was small (smaller than the difference between small and large inputs). To try to make it more difficult for the predictor to generalize, we tried using different architectures for the two networks, multiplying the input by a constant for one of the networks or both, multiplying the network weights by a constant, and changing how many times each state is trained on. Again, none of these approaches seemed to work.

Results with a correctly working RND would have been nice to have since they would clearly build off of prior work in E-MCTS [29], but after extensive RND tuning while being no closer to something that works reliably, we were forced to move on. We considered which kind of source could be trusted absolutely, such that its effect could be isolated and easily analyzed. We arrived at hash-based exploration [43] powered by different hashing algorithms for different behaviour, namely SimHash and a conventional locality-insensitive hash.

SimHash is a known technique, but for the locality-insensitive hash we had to develop new methods. The idea to use LCGs to hash tensors (states) is not new [16], although it has not been used as a source of epistemic uncertainty. We improve on the existing algorithm (hereafter LCGHash) by first taking the element-wise product with a constant tensor, and then reinterpreting the floating point numbers as integers, before using the LCG algorithm. The constant tensor was initialized with random values drawn from a uniform distribution with bounds -100 and 100. Without the random tensor there was an undesirable correlation in the hashes for early states. To avoid collisions, we used a massive hashset with one bit per entry (allowing for only binary outputs, but saving a lot of memory).

3.2 Exploration - Exploitation Trade-off

We use reanalyze [38] in tandem with self-play to generate targets, but only reanalyze generates targets from the data which comes from directed exploration. The portion of self-play and reanalyze data is a hyper-parameter. Using more self-play data means the agent learns from more recent data, from the most "critical" trajectories, those, which are close to what the agent thinks is best. Using more reanalyze data means better sample efficiency, better retention of old information, as well as learning from exploratory data. Running planning during episodes in self-play and running planning for reanalyze is identical from a computation cost, so they both generate fresh targets at the same pace. The catch is that, without self-play, reanalyze is useless, because there is only so much information to be gained from looking at the same positions many times. In our experiments we use an equal portion of self-play targets and reanalyze targets in each training batch. This equal ratio is also reflected in the amount of computation we dedicate to each system.

E-PUCT provides exploratory planning that might not be desirable in the context of two-player, zero-sum games. If we were to use E-PUCT throughout the whole tree, we would be pretending that the adversary also wants to explore, but we should actually expect them to exploit. As such it makes more sense to use E-PUCT only for the current agent's move, and PUCT for the opponent's turns.

PUCT (and by extension E-PUCT) begins by choosing actions based on the prior policy and slowly transitions to prioritizing the value of each action. This behaviour is very nice when the policy is good and when we want the agent to exploit. The problem is when we

want to explore: Exploration should help find actions the policy might otherwise miss. This makes PUCT a subpar choice for exploration because it will overvalue the prior policy and make it difficult to plan away from it.

A natural alternative is UCT [22] and E-UCT, the precursors to PUCT which do not use the prior policy. The benefit of UCT is that it visits actions much more evenly, and is able to optimise for the desired value well, but the fact that it visits actions so evenly is also its downside. We have a limited search budget, and the broader we explore, the shorter the planning horizon beyond which we cannot see. Using UCT also means we throw away the prior policy which is a bad idea since the policy is a helpful tool for pruning uninteresting actions.

Sequential Halving [20] seems like the perfect choice as a trade-off between being selective (to search deeply), but also visiting broadly enough at the start to see interesting actions for exploration. Sequential Halving was successfully used in Gumbel MuZero [12], and in our experiments we use it as well, although we choose to sample a much larger number of actions so that we are able to explore further away from the policy.

3.3 Targets

We have to be careful about the targets we extract in reanalyze. Consider that in exploratory episodes we are intentionally going into areas where the agent is uncertain, so predictions there are unreliable. If we use TD-learning targets which take the "raw" output from a state along this trajectory, it is likely to have a large error. We would also be approximating the value of the exploratory policy $v_{\pi_{explore}}(s)$ (produced when we select actions according to the optimistic value). To minimize these errors, we choose to use the discounted action-value $v_{target}(s) = \gamma q(s, a)$ of the action that the exploitatory policy would have picked after planning.

The UBE target we use reflects the type of exploration we want and expect. We acknowledge that we will never fully explore the state space, and we cannot afford to waste resources exploring in every direction, so we disregard those actions for which we already have a negative preconception. This is echoed in the type of UBE targets which we give: The target is the uncertainty of the action we would have chosen if we had followed the exploratory policy $u_{target}(s) = \gamma^2 \sigma^2_{mcts}(s')$. This means the UBE target follows the uncertainty of the target we would actually choose during exploration.

4 Evaluation

We compare how the two sources of evaluation generalize throughout training in Figure 1. SimHash reports that most `early` positions were seen almost immediately, even though these states are not included in the dataset. This shows SimHash generalizes well over the early states. The `late` states are more unique, so as expected, we do not see as much of an increase. LCGHash should not generalize, and that is what we see: both `early` and `late` remain close to zero, since they are filtered out of the training data. Based on this we can conclude that SimHash and LCGHash behave as designed: SimHash generalizes, while LCGHash does not.

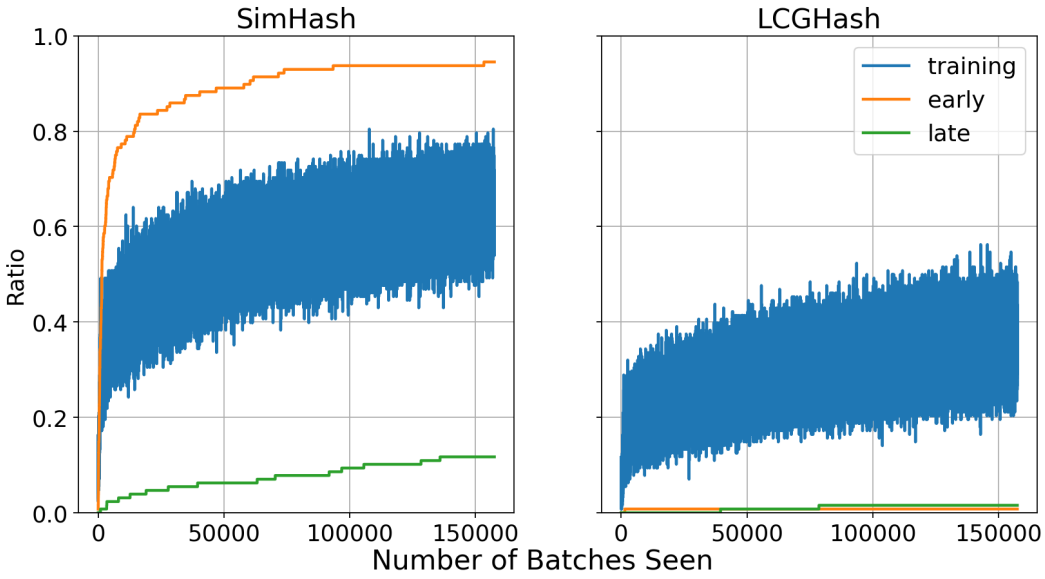

Figure 1: Ratio of positions in a batch that are "seen" throughout training. Each line refers to a different kind of batch: `training` is the batch being put into the hash set. `early` is a batch of early positions (ply = 8), and `late` is a batch of late positions (ply = 60). Both `early` and `late` were filtered out from the dataset, so the pictured increase is directly because of hash-collisions.

We compare how much of the state space is reported as seen by the two hashes in Figure 2. We generate random states at different depths, and then check whether the two hashes have seen these states. LCGHash starts seeing unseen states almost immediately, while for SimHash the early states are consistently seen. If we use LCGHash to explore, we would still be exploring the many unseen early game states. On the other hand, further exploration with SimHash would be focused on later states, since that is where the uncertainty still is. The behaviour of LCGHash undesirable, because after just a few moves, all states are new. When faced with this situation, directed exploration fails, because all directions look equally new and interesting. This makes LCGHash a bad choice for environments with a large state space and branching factor.

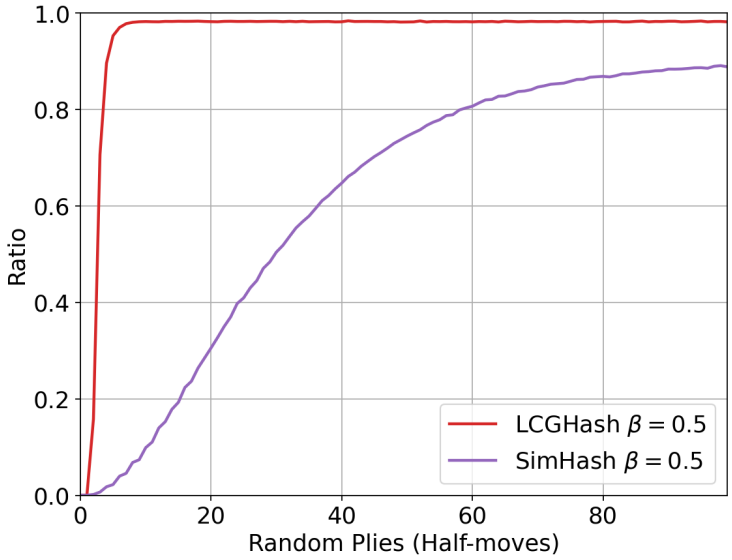

Figure 2: The ratio of unseen states at different depths along random trajectories.

We examine the effect of directed exploration as well as the exploration parameter β on the uniqueness of visited states throughout training in Figure 3. Directed exploration increases the ratio of unique states, and higher values of β increase it even further. There is no noticeable difference when SimHash or LCGHash is used.

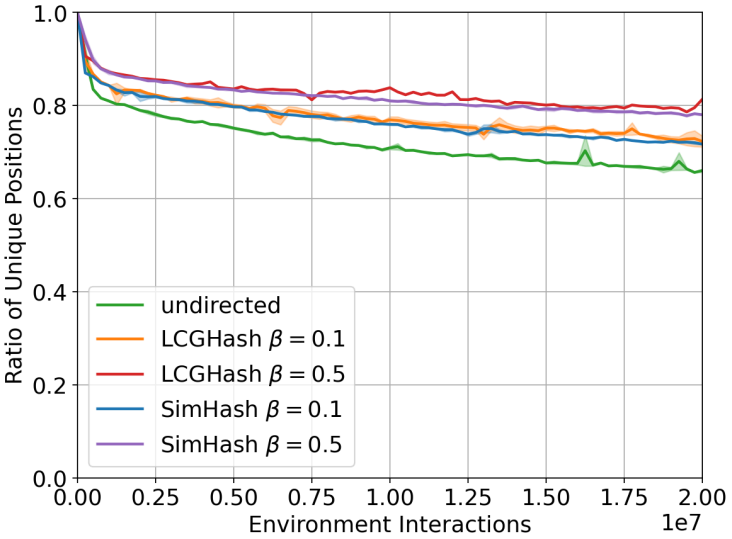

Figure 3: Ratio of unique states (compared to the growing replay buffer) in the last 250k interactions at each point during training.

Finally, we look at the strength of the resulting models in Figure 4. The rating is determined by playing many matches between each checkpoint and then using `bayeselo` [10]

to compute Elo ratings which best predict the match outcomes. All agents achieve similar Elo rating despite different sources of uncertainty and kinds of exploration. This shows that the kind of exploration we are driving with E-MCTS and these sources of uncertainty is not beneficial, and in the case of LCGHash with $\beta = 0.5$, might even be detrimental.

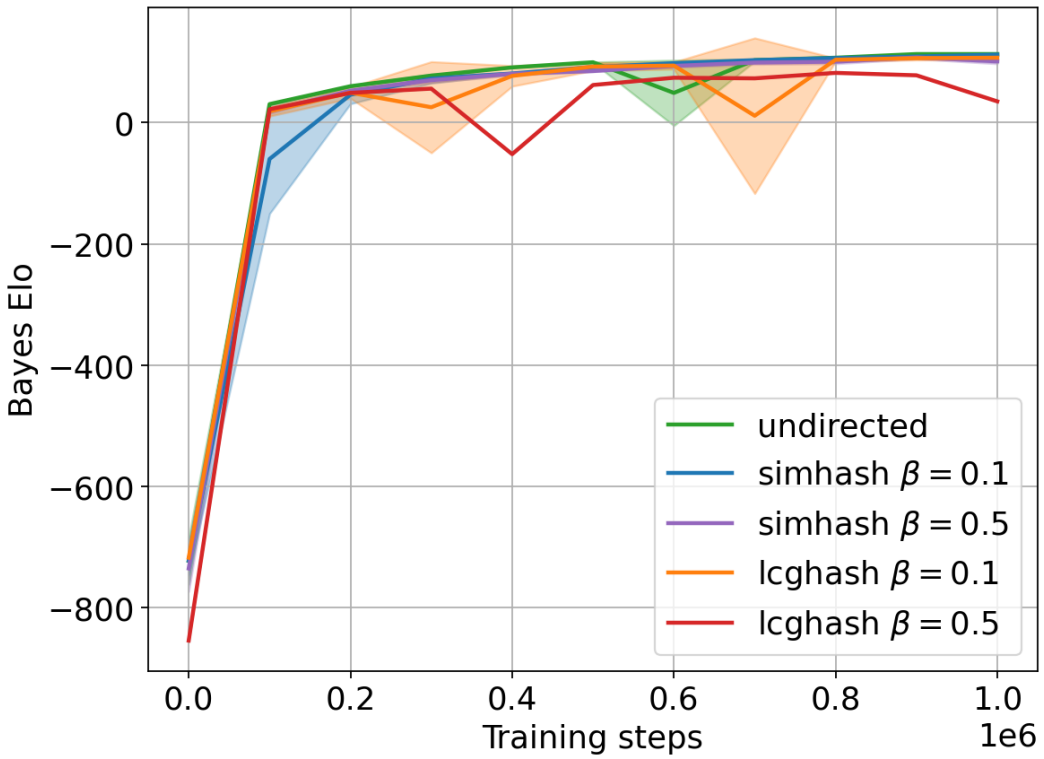

Figure 4: Relative Bayes Elo throughout training. The shaded regions are the standard error across seeds. Each checkpoint played roughly 10k games. The standard deviation reported by `bayeselo` was below 10 for all ratings. Number of seeds per agent: (undirected: 2), (SimHash $\beta = 0.1$: 2), (LCGHash $\beta = 0.1$: 3), (LCGHash $\beta = 0.5$: 1).

5 Responsible Research

We briefly discuss reproducibility and ethical considerations.

The source code has been made public, with instructions for how to run the experiments in the provided `README.md` file. We used a university cluster with 21 Graphical Processing Units (GPUs) working continuously for around two days per seed. It is possible to seed the random number generation of each process, but as soon as the system becomes distributed, the inter-process communication introduces uncontrolled random behaviour. We throttle each process as necessary to maintain a precise ratio of environment interactions to network training steps.

Training of these larger reinforcement learning models requires significant compute which uses much electricity, and so it has an environmental impact. Furthermore, by using a shared

university cluster to perform computations, we delayed the experiments of other researchers.

6 Conclusion

We described the challenges associated with tuning RND [6] and developed LCGHash as a foil for SimHash. We analyzed the effect of directed exploration with these two hash-based sources of uncertainty and saw that although E-MCTS induced additional exploration (as shown in Figure 3) with both sources, neither had an effect on the strength (Elo rating) of the resulting agents. We compared the generalization of the two sources as well as how much of the state space they report as seen.

Just because the local uncertainty sources that we tried did not improve playing strength does not mean that some other source will not benefit. An aspect of uncertainty sources which is perhaps the most interesting is the way the source generalizes. SimHash provides an arbitrary generalization, but the variance of a bootstrapped value-ensemble [31] might provide a more useful source, as a disagreement between good agents would indicate that the state is unclear and we might be able to benefit from exploring it.

It is possible that the agents we trained have not exhausted the "free" novelty that is reachable by undirected exploration, and that with longer training runs, directed deep-exploration agents might benefit once undirected exploration exhausts the close-by novelty.

In self-play, the agent faces a copy of itself. That means that it has a good chance of seeing a positive reward (winning) even without exploration. If we used a much stronger adversary, exploration might be useful in finding a way to beat that specific opponent, without having to cheat by using the other agent's evaluation like in victim-play [45].

There were many other design choices and hyper-parameters which we were not able to explore fully, and each one could induce a benefit from the additional exploration. We did show an important finding though: a truly epistemic source of uncertainty such as LCGHash is not useful when the state space and branching factor is large, because at some depth, all states look new. Future work should only consider generalizing sources of uncertainty when dealing with these complex environments.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[2] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

[3] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

[4] 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.

[5] Sébastien Bubeck, Rémi Munos, and Gilles Stoltz. Pure exploration in multi-armed bandits problems. In *Algorithmic Learning Theory: 20th International Conference, ALT 2009, Porto, Portugal, October 3-5, 2009. Proceedings 20*, pages 23–37. Springer, 2009.

[6] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

[7] Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388, 2002.

[8] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

[9] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

[10] Rémi Coulom. Bayesian Elo Rating. `https://www.remi-coulom.fr/Bayesian-Elo/`, 2005. [Online; accessed 02-05-2024].

[11] Johannes Czech, Patrick Korus, and Kristian Kersting. Improving alphazero using monte-carlo graph search. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, pages 103–111, 2021.

[12] Ivo Danihelka, Arthur Guez, Julian Schrittwieser, and David Silver. Policy improvement by planning with gumbel. In *International Conference on Learning Representations*, 2021.

[13] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[14] Sylvain Gelly and David Silver. Monte-carlo tree search and rapid action value estimation in computer go. *Artificial Intelligence*, 175(11):1856–1875, 2011.

[15] Jean-Bastien Grill, Florent Altché, Yunhao Tang, Thomas Hubert, Michal Valko, Ioannis Antonoglou, and Rémi Munos. Monte-carlo tree search as regularized policy optimization. In *International Conference on Machine Learning*, pages 3769–3778. PMLR, 2020.

[16] Mateen Ulhaq https://stackoverflow.com/users/365102/mateen-ulhaq. How to hash a PyTorch tensor. `https://stackoverflow.com/revisions/77213071/4`, 2023. [Online; accessed 01-05-2024].

[17] Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Mohammadamin Barekatain, Simon Schmitt, and David Silver. Learning and planning in complex action spaces. In *International Conference on Machine Learning*, pages 4476–4486. PMLR, 2021.

[18] Eyke Hüllermeier and Willem Waegeman. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Machine learning*, 110(3):457–506, 2021.

[19] Albin Jaldevik. General tree evaluation for alphazero. 2024.

[20] Zohar Karnin, Tomer Koren, and Oren Somekh. Almost optimal exploration in multi-armed bandits. In *International conference on machine learning*, pages 1238–1246. PMLR, 2013.

[21] Donald Ervin Knuth et al. *The art of computer programming*, volume 3. Addison-Wesley Reading, MA, 1973.

[22] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[23] Wouter Kool, Herke Van Hoof, and Max Welling. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. In *International Conference on Machine Learning*, pages 3499–3508. PMLR, 2019.

[24] Tor Lattimore and Csaba Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020.

[25] Owen Lockwood and Mei Si. A review of uncertainty for deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 18, pages 155–162, 2022.

[26] Gaurav Manek and J Zico Kolter. The pitfalls of regularization in off-policy td learning. *Advances in Neural Information Processing Systems*, 35:35621–35631, 2022.

[27] Thomas M Moerland, Joost Broekens, and Catholijn M Jonker. Efficient exploration with double uncertain value networks. *arXiv preprint arXiv:1711.10789*, 2017.

[28] Nikolay Nikolov, Johannes Kirschner, Felix Berkenkamp, and Andreas Krause. Information-directed exploration for deep reinforcement learning. *arXiv preprint arXiv:1812.07544*, 2018.

[29] Yaniv Oren, Matthijs TJ Spaan, and Wendelin Böhmer. E-mcts: Deep exploration in model-based reinforcement learning by planning with epistemic uncertainty. *arXiv preprint arXiv:2210.13455*, 2022.

[30] Ian Osband, John Aslanides, and Albin Cassirer. Randomized prior functions for deep reinforcement learning. *Advances in Neural Information Processing Systems*, 31, 2018.

[31] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. *Advances in neural information processing systems*, 29, 2016.

[32] Ian Osband, Zheng Wen, Seyed Mohammad Asghari, Vikranth Dwaracherla, Morteza Ibrahimi, Xiuyuan Lu, and Benjamin Van Roy. Epistemic neural networks. *Advances in Neural Information Processing Systems*, 36, 2024.

[33] Georg Ostrovski, Marc G Bellemare, Aäron Oord, and Rémi Munos. Count-based exploration with neural density models. In *International conference on machine learning*, pages 2721–2730. PMLR, 2017.

[34] Brendan O'Donoghue, Ian Osband, Remi Munos, and Volodymyr Mnih. The uncertainty bellman equation and exploration. In *International conference on machine learning*, pages 3836–3845, 2018.

[35] Stephen K. Park and Keith W. Miller. Random number generators: good ones are hard to find. *Communications of the ACM*, 31(10):1192–1201, 1988.

[36] Christopher D Rosin. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230, 2011.

[37] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[38] Julian Schrittwieser, Thomas Hubert, Amol Mandhane, Mohammadamin Barekatain, Ioannis Antonoglou, and David Silver. Online and offline reinforcement learning by planning with a learned model. *Advances in Neural Information Processing Systems*, 34:27580–27591, 2021.

[39] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[40] Aleksandrs Slivkins et al. Introduction to multi-armed bandits. *Foundations and Trends® in Machine Learning*, 12(1-2):1–286, 2019.

[41] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3:9–44, 1988.

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

[43] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

[44] US Tak Association. How To Play The Beautiful Game of Tak. `https://ustak.org/play-beautiful-game-tak/`, 2021. [Online; accessed 23-06-2024].

[45] Tony Tong Wang, Adam Gleave, Tom Tseng, Kellin Pelrine, Nora Belrose, Joseph Miller, Michael D Dennis, Yawen Duan, Viktor Pogrebniak, Sergey Levine, et al. Adversarial policies beat superhuman go ais. In *International Conference on Machine Learning*, pages 35655–35739. PMLR, 2023.

[46] Tom Zahavy, Vivek Veeriah, Shaobo Hou, Kevin Waugh, Matthew Lai, Edouard Leurent, Nenad Tomasev, Lisa Schut, Demis Hassabis, and Satinder Singh. Diversifying ai: Towards creative chess with alphazero. *arXiv preprint arXiv:2308.09175*, 2023.