Monte Carlo Tree Search in the Presence of Transition Uncertainty

Farnaz Kohankhaki[1*], **Kiarash Aghakasiri**[1, 2*], **Hongming Zhang**[1], **Ting-Han Wei**[1], **Chao Gao**[2], **Martin Müller**[1]

[1]University of Alberta
[2]Edmonton Research Center, Huawei Canada
{kohankha, aghakasi, hongmin2, tinghan, mmueller}@ualberta.ca, chao.gao4@huawei.com

Abstract

Monte Carlo Tree Search (MCTS) is an immensely popular search-based framework used for decision making. It is traditionally applied to domains where a perfect simulation model of the environment is available. We study and improve MCTS in the context where the environment model is given but imperfect. We show that the discrepancy between the model and the actual environment can lead to significant performance degradation with standard MCTS. We therefore develop Uncertainty Adapted MCTS (UA-MCTS), a more robust algorithm within the MCTS framework. We estimate the transition uncertainty in the given model, and direct the search towards more certain transitions in the state space. We modify all four MCTS phases to improve the search behavior by considering these estimates. We prove, in the corrupted bandit case, that adding uncertainty information to adapt UCB leads to tighter regret bound than standard UCB. Empirically, we evaluate UA-MCTS and its individual components on the deterministic domains from the MinAtar test suite. Our results demonstrate that UA-MCTS strongly improves MCTS in the presence of model transition errors.

1 Introduction

The Monte Carlo Tree Search (MCTS) framework (Browne et al. 2012) approaches sequential decision-making problems by selective lookahead search. It manages the balance of exploration and exploitation with techniques such as UCT (Kocsis, Szepesvári, and Willemson 2006). Often combined with machine learning, it has been enormously successful in both games (Silver et al. 2016; Gao, Müller, and Hayward 2018; Gao 2020; Saffidine 2008; Nijssen and Winands 2010) and non-game applications (Lu et al. 2016; Mansley, Weinstein, and Littman 2011; Sabharwal, Samulowitz, and Reddy 2012; Cazenave 2010). In these applications, a perfect simulation model allows for efficient lookahead search. However, in many practical applications, only an imperfect model is available to the agent. Yet lookahead using such a model can still be useful. We improve MCTS for this setting.

One research area that studies imperfect models of the environment is model-based reinforcement learning (MBRL). Here, an agent builds its own model through limited real world interactions. The resulting learned model, when used for lookahead search, can either be for planning or for producing more accurate training targets (Silver, Sutton, and Müller 2008). It can also be used to generate simulated training samples for better sample efficiency (Sutton and Barto 2018). The learned model may be inaccurate for many reasons, including stochasticity of the environment, insufficient training, insufficient capacity, non stationary environments, etc. Consequently, there is a rich body of research on *uncertainty* in MBRL (Abbas et al. 2020; Xiao et al. 2019; Buckman et al. 2018).

While previous approaches to using search with imperfect models exist (Vemula et al. 2020; Vemula, Bagnell, and Likhachev 2021), to the best of our knowledge, there is no prior work that directly adapts MCTS to deal with model uncertainty. In our work, we define transition uncertainty as a measure of difference between the state transitions in the perfect model and in the model that is available to the agent. We use a neural network to estimate this uncertainty.

Our Uncertainty Adapted MCTS (UA-MCTS) approach implements the main components of the MCTS framework in a way that guides the search away from states with high uncertainty. We compare the performance of our proposed methods with MCTS baselines in three deterministic MinAtar environments (Young and Tian 2019). In each case the search agent "believes" it is playing the real game. However, the rules of the game itself have changed, and the agent only learns about this change slowly when it acts in the real environment. The results show that UA-MCTS is able to outperform the baseline MCTS with an imperfect model.

Our approach is inspired by the work of (Vemula et al. 2020) where a robotic arm has to solve tasks despite being handicapped, e.g. by a broken motor or by an unmodeled weight restriction. To show how an agent should adapt UCB-based exploration strategy in the presence of environment uncertainties, we first consider a case of stochastic bandits (Lattimore and Szepesvári 2020) along with corrupted feedback. We prove that incorporating uncertainty information can enhance the performance of UCB, yielding a regret bound that is more constrained compared to the standard UCB. We also prove that in the general case of tree search, with similar modification of UCT, our UA-MCTS approach maintains its completeness property, ensuring that as the number of iterations goes to infinity, all nodes will be

[*]These authors contributed equally.

consistently explored. To further motivate our approach, we compare the scenarios of learning to improve the transition function, using MCTS, directly against the easier task of just learning a transition uncertainty function with UA-MCTS. In both cases, learning occurs online; the former is used with MCTS while the latter is used with UA-MCTS. Our results show that learning the transition function is much harder than learning transition uncertainty, which justifies the use of UA-MCTS in such settings.

2 Background

2.1 The Imperfect Model Problem Setting

We focus on *deterministic* Markov Decision Processes (MDP) which can be expressed as a 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, M, \gamma)$ with state space $\mathcal{S}$, action space $\mathcal{A}$, rewards $\mathcal{R}$ which map an (s, a) pair to a scalar reward, deterministic dynamics $s_{t+1} = M(s_t, a)$, and a discount factor $0 \leq \gamma \leq 1$. An agent *policy* maps each state in $\mathcal{S}$ to an action in $\mathcal{A}$. The goal is to find a policy that maximizes the total discounted reward.

During the course of a search, the agent only has access to an *imperfect model* $\hat{M}(s, a)$ of the environment. We assume that $\hat{M}$ is both given and fixed within one search. This is in contrast to the model learning case where $\hat{M}$ is neither given nor fixed. Each search is used to select an action, which is then executed in the "real world" M.

Both M and $\hat{M}$ take a state and an action as input and return the next state. However, the next state predicted by $\hat{M}$ might be incorrect. After each search in $\hat{M}$, the agent decides on an action a, which is then executed in the "real world" M, and observes the next true state. $\hat{M}$ and M share the same $\mathcal{S}$, $\mathcal{A}$, $\mathcal{R}$, and γ.

We model the *transition uncertainty* $U(s, a)$ as an (arbitrary) function of the difference between the next-state predictions made by $\hat{M}$ and M for the same state-action pair (s, a). Like M, U is not directly available to the agent, thus it learns an estimation of $U(s, a)$, noted as $\hat{U}(s, a)$, during interactions with the environment. A special case is that when the MDP contains only one state, the problem becomes a *corrupted K-armed stochastic bandit* where there are K actions. In this case, the *transition uncertainty* leads to corrupted reward distribution — the agent takes arm i, instead of receiving from the reward from real distribution p_i, a reward is sampled from corrupted distribution $\hat{p}_i$. Following standard bandit assumptions, we use $\mu_i \in [0, 1]$ and $\hat{\mu}_i \in [0, 1]$ to denote the expected rewards for p_i and $\hat{p}_i$, respectively. The difference is noted as $\delta_i = |\mu_i - \tilde{\mu}_i|$.

2.2 Monte Carlo Tree Search (MCTS)

MCTS conducts a selective tree search by repeating four steps (Algorithm 1): An in-tree *selection* method such as UCT (Kocsis, Szepesvári, and Willemson 2006) descends along a path in the tree until a leaf node is reached, which is then *expanded*. A *simulation* evaluates the selected leaf node. Finally, the nodes along the selection path, up to and including the root of the tree, are updated according to the evaluation result during *backpropagation* (Browne et al. 2012). Each of the four steps of MCTS are modified in UA-MCTS.

Algorithm 1: MCTS Framework

```
function MCTS($s_0$)
    create a root node $v_0$ with state $s_0$
    for $N_I$ do
        $v_s \leftarrow$ SELECT($v_0$)
        if $N(v_s) > 0$ then
            $v_s \leftarrow$ EXPAND($v_s$)
        end if
        $value \leftarrow$ SIMULATE($S(v_s)$)
        BACKPROPAGATE($v_s$, $value$)
    end for
    $v_{best} \leftarrow$ choose the most visited child of $v_0$
    return $action(v_{best})$
end function
```

We use the following notation: For node v in the current search tree T, $S(v)$ is the feature vector of the state that is represented by v; $N(v)$ is the number of times v has been visited throughout the search; $Q(v)$ is the sum of rewards observed at v; $Par(v)$ is the parent of v in T, with $Par(root) = NULL$; $Ch(v)$ is the current set of v's children in T; and $\hat{U}(v)$ is the estimated transition uncertainty of v which is equal to $\hat{U}(s, a)$, where (s, a) is the transition in T that leads to node v. The choices for $\hat{U}(v)$ are further elaborated on in Subsection 5.1. UA-MCTS is controlled by five hyperparameters: the total number of iterations N_I, the number N_S of simulations to evaluate a leaf node of T, the maximum depth of each simulation D_S, the exploration constant c, and the uncertainty factor τ which controls how much the uncertainty affects the behavior of UA-MCTS.

3 Uncertainty Adapted UCB

Before diving into full MCTS for corrupted MDPs, we first consider the corrupted bandit case. For the real and corrupted environments, we assume the optimal arms are respectively noted as i^* and $\hat{i}^*$. At time t ($t > 1$), let $\hat{x}_{i,(t-1)}$ be the empirical average reward collected for arm i before time t, and $N_i(t-1)$ be the visit count of arm i. Naturally, the agent has to consider uncertainty information δ_i; one sensible idea is to adapt the UCB score function for each arm i, as follows:

$$UA - UCB_i(t) = \hat{x}_{i,t-1} + c\sqrt{\frac{\ln(t-1)}{N_{i,t-1}}} \cdot (1 - \delta_i)$$

Here, $\hat{x}_{i,t}$ is an estimate for $\hat{\mu}_i$, e.g., empirical mean reward.

3.1 Regret Bound

We provide a regret bound for the UA-UCB strategy for corrupted bandits (proof shown in Appendix C (Kohankhaki et al. 2023)).

Theorem 1.

$$R_n \leq \sum_{i=1}^{k} \Big[\frac{4\beta_i}{\hat{\Delta}_i^2} + C\Big](\hat{\Delta}_i + \delta_i + \mu_{i^*} - \hat{\mu}_{\hat{i}^*})\ln(n-1)$$

Here C is a constant, $\beta_i = c^2(1-\delta_i)^2$, $\Delta_i = \mu_{i^*} - \mu_i$ and $\hat{\Delta}_i = \hat{\mu}_{i^*} - \hat{\mu}_i$, assuming $0 \leq \delta_i \leq 1 - \sqrt{\frac{1}{2c^2}}$.

Clearly, if $\delta_i = 0$, this regret bound matches UCB for stochastic bandits. If $\delta_i \neq 0$, we can also see that, acting using UCB resulting into very similar regret notation except that $\beta_i = c^2$ (see Appendix C (Kohankhaki et al. 2023)). Since $c^2(1-\delta_i)^2 \leq c^2$, we see that UA-UCB, by considering the uncertainty information in selection, produces a tighter bound. Note that when $\delta_i > 1 - \sqrt{\frac{1}{2c^2}}$, both UCB and UA-UCB lead to linear regret, i.e., the reward observed by agent is too much misleading w.r.t the real environment such that optimistic strategies based on estimation of $\hat{p}$ become not beneficial.

4 Uncertainty Adapted Monte Carlo Tree Search (UA-MCTS)

How can search in an imperfect model $\hat{M}$ improve decision-making in M? Of course, the optimal strategy for the agent is to exploit the mathematical relation between $\hat{M}$ and M; however, this information is unknown for the agent. In UA-MCTS, we adapt MCTS to behave more conservatively in states where the estimated uncertainty is larger. By doing so, we discourage, but do not completely give up on, searching through the more uncertain parts of the model.

Guided by the insights from the corrupted bandit case, in UA-MCTS, we use four custom functions for *selection*, *expansion*, *simulation*, and *backpropagation* within an MCTS framework which utilizes this learned transition uncertainty. These four functions are described in subsections 4.1 - 4.4.

4.1 UA-Selection

Similar to UA-UCB, in UA-Select, the exploration term in the standard UCT formula is dampened by a multiplicative term $1 - \alpha_i$, so that children with higher uncertainty will be explored less. α_i is a softmax of $\hat{U}$ with a temperature parameter set to the uncertainty factor $\tau > 0$. A lower τ makes the algorithm more sensitive to the predicted uncertainty. With a higher τ UA-Selection behaves more like the baseline MCTS selection. Algorithm 2 describes the modified *selection* function in pseudo-code.

We prove that with this modification of UCT, our UA-MCTS remains complete in the sense that all nodes will be visited continuously in the limit as N_I goes to infinity. We prove this by induction.

Lemma 1. *$\forall v \in Tree$, if $\lim_{N_I\to\infty} N(v) = \infty \Rightarrow \forall v_i \in Ch(v) \lim_{N_I\to\infty} N(v_i) = \infty$.*
(Proof in Appendix D (Kohankhaki et al. 2023))

Theorem 2. *(Completeness) If $N_I \to \infty$, then $\forall v \in Tree, N(v) \to \infty$*

Algorithm 2: Uncertainty Adapted Selection Algorithm.

```
Parameter: temperature $\tau$ for softmax
function SELECT($v$)
    while $v$ is expanded do
        for $v_i \in Ch(v)$ do
            $\alpha_i \leftarrow \frac{e^{\hat{U}(v_i)/\tau}}{\sum_{v_j \in Ch(v)} e^{\hat{U}(v_j)/\tau}}$
        end for
        $v \leftarrow \operatorname{argmax}_{v_i \in Ch(v)} \frac{Q(v_i)}{N(v_i)} + c\sqrt{\frac{\ln N(v)}{N(v_i)}} \cdot (1 - \alpha_i)$
    end while
    return $v$
end function
```

Proof. We prove this by induction on depth of the nodes in the tree.

1. Induction base: $N(root) \to \infty$.
2. Induction step: $\lim_{N_I\to\infty} N(v) = \infty. \Rightarrow \forall v_i \in Ch(v), \lim_{N_I\to\infty} N(v_i) = \infty$ (Proved in Lemma 1).

4.2 UA-Expansion

To discourage searching parts of the tree where the model is more inaccurate, UA-Expansion gives children with higher uncertainty a lower chance to be added to the tree. While expanding node v, we initialize the uncertainty of each child v_i based on the uncertainty of the transition $(S(v), a_i)$ that yielded v_i and store it as $\hat{U}(v_i)$. This stored value can then be accessed by other parts of the UA-MCTS algorithm. Next, with probability $1 - \tau/10$ we eliminate exactly one child from the tree. The choice of which child to eliminate is dependent on probabilities that are proportional to each child node's uncertainty. The uncertainty factor τ is used in UA-Expansion as the elimination probability $\tau/10$; the scaling constant $1/10$ was chosen empirically. Algorithm 3 shows the pseudo-code for UA-Expansion.

4.3 UA-Simulation

In regular MCTS simulation, the returned value is the average of the N_S rollouts. To adapt simulation to transition uncertainty, we weigh each result based on a measure of the uncertainty of the whole rollout trajectory. Assume T_i is the rollout trajectory that the ith rollout took and $\mathcal{T}$ is the set of rollout trajectories. We define the uncertainty $\sigma(T_i)$ of trajectory $T_i = (s_1, a_1, s_2, a_2, \cdots, s_h, a_h, s_{h+1})$ as

$$\sigma(T_i) \doteq \sum_{k=1}^{h} \gamma^{k-1}\hat{U}(s_k, a_k)$$

The weight α_i for rollout i is defined as the softmax

$$\alpha_i \doteq \frac{e^{-\sigma(T_i)/\tau}}{\sum_{j=1}^{N_S} e^{-\sigma(T_j)/\tau}}$$

Algorithm 3: Uncertainty Adapted Expansion Algorithm

```
Parameter: probability $\tau$ for deleting a node
function EXPAND($v$)
    for $a_i \in \mathcal{A}$ do
        $s_i, r_i \leftarrow \hat{M}(S(v), a_i)$
        create a node $v_i$ with state $s_i$ and reward $r_i$
        $\hat{U}(v_i) \leftarrow \hat{U}(S(v), a_i)$
        $N(v_i) \leftarrow 0$
        $Q(v_i) \leftarrow 0$
    end for
    $x \sim Uniform(0, 1)$
    if $\sum_{v_j \in Ch(v)} \hat{U}(v_j) > 0$ and $x < (1 - \tau/10)$ then
        for $a_i \in \mathcal{A}$ do
            $\alpha_i \leftarrow \frac{\hat{U}(v_i)}{\sum_{v_j \in Ch(v)} \hat{U}(v_j)}$
        end for
        choose node $v_i$ with probability $\alpha_i$
        delete node $v_i$
    end if
    return a random child of $v$
end function
```

Again, a lower τ makes α_i more sensitive to the predicted uncertainties. Let g_i be the sum of discounted rewards for rollout i. Then the weighted average return G of all N_S simulations is

$$G \doteq \sum_{i=1}^{N_S} \alpha_i \cdot g_i$$

Pseudo-code for UA-Simulation is presented in Appendix B (Kohankhaki et al. 2023).

4.4 UA-Backpropagation

In Algorithm 4 UA-Backpropagation, nodes with more certainty have more impact on their parents. To do that, a child's value is modified with a multiplicative term that is based on its uncertainty before it is used to update its parent's value. This multiplicative term is the softmax of $-\hat{U}$ with a temperature parameter set to the uncertainty factor τ, to assign children with lower uncertainty a higher weight.

5 Experiments

We experimentally test the performance of UA-MCTS.[1] In each test, we define a specific transition uncertainty U and a method for learning its approximation $\hat{U}$. We compare UA-MCTS with baseline MCTS in three modified MinAtar environments (Young and Tian 2019). For each of the deterministic MinAtar environments – Space Invaders, Freeway, and Breakout – we briefly explain the modified domain, and discuss the experimental results. Also, we compare learning to improve the transition function with learning the uncertainty model in a toy environment setting. Moreover, in Appendix A (Kohankhaki et al. 2023), we discuss the scenario where all paths to the goal contain uncertainty.

[1]https://github.com/ualberta-mueller-group/UAMCTS

Algorithm 4: Uncertainty Adapted Backpropagation Algorithm.

```
Parameter: uncertainty factor $\tau$
function BACKPROPAGATE($v$, $value$)
    while $v$ is not NULL do
        $N(v) \leftarrow N(v) + 1$
        $\alpha = \frac{e^{-\hat{U}(v)/\tau}}{\sum_{n \in Ch(Par(v))} e^{-\hat{U}(n)/\tau}}$
        $Q(v) \leftarrow Q(v) + \alpha \cdot value$
        $value \leftarrow value \cdot \gamma$
        $v \leftarrow Par(v)$
    end while
end function
```

5.1 Defining Uncertainty and Learning its Estimate

The uncertainty estimate U and the implementation of its approximation $\hat{U}$ can be chosen freely in our framework. For our experiments we use the following approach: Given M and $\hat{M}$, we define the *transition uncertainty* U as the squared difference of the state vectors

$$U(s, a) = \big(\hat{M}(s, a) - M(s, a)\big)^2.$$

A neural network uncertainty model $\hat{U}$ which approximates U is learned by regression. After each action in the environment M, $U(s, a)$ is computed and the tuple $\langle s, a, U(s, a)\rangle$ is added to a buffer B. After every I steps in the environment, the neural network $\hat{U}$ is trained for E training steps. τ decays exponentially with the update formula $\tau = \tau/10$ at each I steps in the environment. This scheduling scheme for τ enables UA-MCTS to be more sensitive to the predicted uncertainty. Each training step performs a gradient descent update with a randomly selected batch $B' \subset B$ and loss function L:

$$L = \sum_{s,a,U(s,a) \in B'} \Big(\hat{U}(s, a) - U(s, a)\Big)^2$$

For each real world action decision, UA-MCTS is run with the current state as a start state for N_I iterations. Whenever a node v is added to the tree from transition (s, a), we define its uncertainty as $\hat{U}(v) = \hat{U}(s, a)$. In the offline case, $\hat{U} = U$; in the online case, $\hat{U}$ is learned as above.

5.2 Setup of Experiments

We test our method on the three deterministic games Space Invaders, Freeway and Breakout in the MinAtar framework (Young and Tian 2019). While the original games are all continuing tasks, we used an episodic version for our experiments.

We modify each game to be "slightly broken" by changing the transition function M. The agent still believes it is playing the original game and plans using that model $\hat{M}$, which is no longer correct for the modified game. We compare UA-MCTS and its four components with two baseline MCTS algorithms in this setting. For each game, we study two uncertainty scenarios and compare a total of seven search algorithms:

1) In the *offline scenario*, the agent has access to the true uncertainty U of the model, but not to the perfect model M itself. This scenario evaluates the performance of UA-MCTS with a "perfect uncertainty" model. The reason for this experiment is to separate out the difficulty of not knowing M from the extra difficulty due to the training errors in the estimate $\hat{U}$.

2) In the *online scenario*, the agent does not have access to U, and therefore has to learn an approximation $\hat{U}$ online from real experience. The agent starts with a random initialization of the uncertainty network and collects the buffer of transitions B that is used to train $\hat{U}$ as explained in Section 5.1.

The seven search algorithms tested are as follows: the two baselines use standard MCTS without any UA modifications.

1) "True Model" MCTS is allowed to use M in its search. It is shown as a horizontal dashed green line in the figures.

2) "Corrupted Model" MCTS uses $\hat{M}$ for all its planning, and is shown as a horizontal dashed red line in the figures.

3-7) All versions of UA-MCTS work as follows: they use $\hat{M}$ for all their planning. In the offline scenario, they use the true U from the beginning, as explained above. In the online scenario, they start by running the Corrupted Model MCTS (no UA modifications), and gain real experience from the moves played. The UA-MCTS versions labeled "Backpropagation", "Selection", "Expansion" and "Simulation" use the UA modifications only for this one component of MCTS, and use unmodified MCTS for the other three parts. The "Combined" version uses all four enhancements.

In the offline scenario we show results for all seven algorithms. In the online scenario, for simplicity we only show results for the combined version of UA-MCTS and the two MCTS baselines.

For each combination of game and algorithm we performed a parameter sweep over the exploration constant c from the set $\{0.5, 1, \sqrt{2}, 2\}$. For the "Combined" version in the online scenario, we used the best c found for the offline scenario. In the offline scenario the uncertainty factor τ is set to 0.1 (a small number so that UA-MCTS is more sensitive to the true uncertainty), and in the online scenario the τ is initialized to 10, then decays until it reaches 0.1. The uncertainty model in the online scenario is a fully connected neural network with two hidden layers of 128 hidden units each. The number of training steps E and training frequency I are both 5000, and the step size is 10^{-3} for the Adam optimizer (Kingma and Ba 2014). Table 1 shows a list of other hyperparameters used in UA-MCTS.

5.3 Space Invaders

In Space Invaders, the agent tries to eliminate 24 enemies by shooting at them (Young and Tian 2019). We modified this environment by disabling the shoot action in five out of ten positions, at indices 2, 3, 4, 5, and 6. As shown in Figure 1, the performance of the baseline MCTS drops from an average reward of 18.4 for the True Model MCTS to 16.3 for the Corrupted Model MCTS, where the agent is unaware of its limitations on shooting.

In the offline scenario, UA-MCTS worked surprisingly well, and even exceeded the performance of the True Model MCTS in all configurations except UA-backpropagate-only. As a single modification, UA-expand-only performed best. The combined UA-MCTS achieved an average reward of 23.8, which is close to the perfect play reward of 24. Even in the online setting, after training $\hat{U}$, combined UA-MCTS outperforms the True Model MCTS. In this game, low value states are strongly correlated with high uncertainty states where the agent cannot shoot. Since UA-MCTS discourages visits to these states, its search becomes more efficient than the True Model MCTS despite the imperfect model.

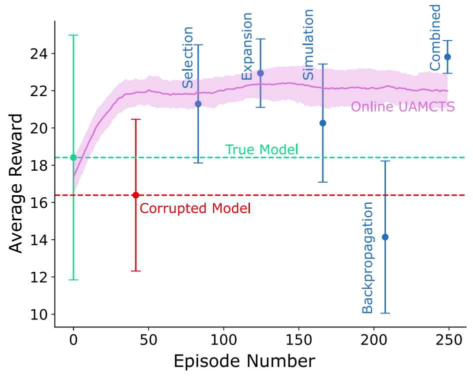

Figure 1: Offline (vertical bars) and online scenarios for Space Invaders. Bars and shaded areas show $mean \pm std$ of rewards over 100 runs for the offline and 15 for the online scenarios. For the corrupted and true models (two-leftmost cases), the best c is 2 and $\sqrt{2}$ respectively. The best c parameter chosen for each of the UA-MCTS algorithms (the remaining five cases) are [2, 2, 1, 0.5, $\sqrt{2}$] from left to right. For the online scenario, we plot the moving average of the reward with a window size of 50.

5.4 Freeway

In Freeway, the agent's goal is to reach the top of the screen without touching enemies along the way (Young and Tian 2019). In the modified game M, executing the action "None" moves the agent up in six out of ten locations. Figure 2 shows that offline, combined UA-MCTS achieves the average reward of 0.9 and outperforms Corrupted Model MCTS. The individual performance of each component is better than the Corrupted Model, with UA-expand-only again performing best. In the online setting, UA-MCTS is able to outperform the Corrupted Model MCTS, but cannot approach the True Model MCTS. Comparing the two results (offline and online) for this game, we conjecture that the uncertainty network's inaccuracies limit the performance.

The online UA-MCTS agent converges to its best perfor-

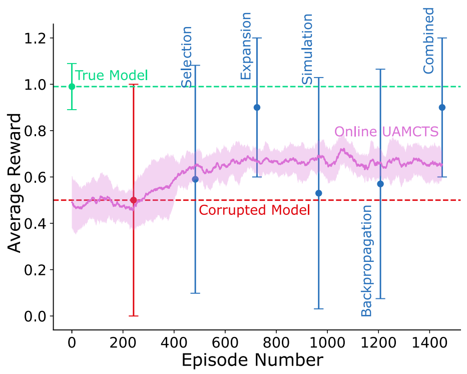

Figure 2: Offline (vertical bars) and online scenarios for Freeway. Bars and shaded areas show $mean \pm std$ of rewards over 100 runs for the offline and 15 for the online scenarios. For the corrupted and true models (two-leftmost cases), the best c is 0.5 and 2 respectively. The best c parameter chosen for each of the UA-MCTS algorithms (the remaining five cases) are [2, 2, $\sqrt{2}$, 2, $\sqrt{2}$] from left to right. For the online scenario, we plot the moving average of the reward with a window size of 50.

mance in Space Invaders using fewer episodes than in Freeway. The reason is that Space Invaders episodes are longer, so the agent can learn from more environment interaction samples.

5.5 Breakout

In Breakout, the agent controls a paddle at the bottom of the screen, which is used to bounce a ball back up. The goal is to hit and destroy as many bricks as possible (Young and Tian 2019). In our modified game, the paddle fails to stop the ball in two out of ten positions, ending the game. To overcome this, the agent must plan ahead to keep the ball in play without using these corrupted positions. Figure 3 shows that UA-MCTS outperformed the Corrupted Model MCTS baseline and performed almost as well as the True Model MCTS. UA-select-only and UA-expand-only also performed very well, while UA-backpropagate-only and UA-simulate-only performed poorly. Another observation is the performance drop in the online setting w.r.t the offline setting, although it still performed better than the Corrupted Model MCTS. Since the main difference between the two settings is the uncertainty, the online setting is likely limited by its learned uncertainty's lower accuracy. The effect of this inaccuracy, and potential improvements, remain an interesting future direction of research.

	Space Invaders	Freeway	Breakout
N_I	10	100	100
D_S	20	50	50
N_S	10	10	10

Table 1: Experiment settings

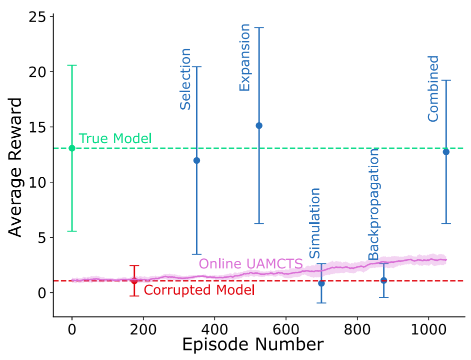

Figure 3: Offline (vertical bars) and online scenarios for Breakout. Bars and shaded areas show $mean \pm std$ of rewards over 100 runs for the offline and 15 for the online scenarios. For the corrupted and true models (two-leftmost cases), the best c is 2 and $\sqrt{2}$ respectively. The best c parameter chosen for each of the UA-MCTS algorithms (the remaining five cases) are [1, 2, $\sqrt{2}$, 2, $\sqrt{2}$] from left to right. For the online scenario, we plot the moving average of the reward with a window size of 50.

5.6 Learning the Uncertainty VS. Learning the Transition Function

An alternative approach to learning the uncertainty, then applying that to UA-MCTS, is to use MCTS with a learned transition function that approximates the true environment. We justify the choice of focusing on the former in this paper as follows. We believe that training the transition function is generally more difficult than training the uncertainty because state representations tend to be more complex. In contrast, uncertainty can be represented with a single scalar value regardless of the environment. As an example, the feature vector representing each state contains 306, 34, and 107 scalar elements for Space Invaders, Freeway, and Breakout, respectively. Therefore, with the same amount of resources, we should be able to learn a more accurate uncertainty model than a transition model. (Vemula et al. 2020; Vemula, Bagnell, and Likhachev 2021) also mention the same intuition, which motivated their approach. To illustrate this with an example, we designed a toy environment called the 2-Way GridWorld. A snapshot of this environment is presented in Figure 4. The agent starts at position $(1, 0)$ and at each time step it can choose any of the four actions: "Up", "Down", "Left", and "Right". The terminal state is at position $(1, 6)$. When the agent reaches the position $(1, 6)$, the game terminates and the agent gets a reward of 10. The maximum number of steps in each episode is 50. In the corrupted model the agent does not know that the wall in position $(0, 2)$ exists. Therefore, it assumes that it can reach the goal either from the top path or from the bottom path. The reality is that the top path is not a valid plan.

We compare the two scenarios: 1) Using combined UA-MCTS and learning the uncertainty online: we use combined UA-MCTS and the same method and setup for learning the uncertainty as in the MinAtar environments, except that the uncertainty model is a fully connected neural net-

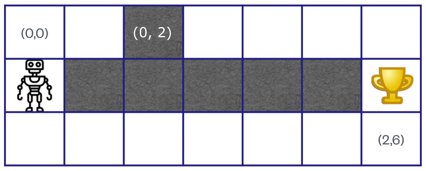

Figure 4: 2-Way GridWorld Environment. White cells are empty and grey cells are walls.

work with no hidden layers. τ is initialized to 10 and decays until it reaches 0.1. 2) Using MCTS and learning the transition function online: We use MCTS and learn the transition function online. The process of learning the transition function is the same as learning the uncertainty model, except that the buffer B is filled with $\langle s, a, M(s,a)\rangle$ samples. For the first I steps, the corrupted model is used to gather a buffer, since the transition function has not been trained yet.

We used three different transition functions: a) a linear regression, b) a fully connected neural network with 1 hidden layer consisting of 8 units, and c) a fully connected neural network with 1 hidden layer with 16 units. All models predict the feature vector of the next state. We also compare with the two baselines "True Model" and "Corrupted Model". We performed a parameter sweep over the exploration constant c from the set $\{0.5, 1, \sqrt{2}, 2\}$ for the "True Model" and "Corrupted Model". We used the best c found for the "Corrupted Model" for the two scenarios. In this set of experiments $N_I = 10$, $N_S = 10$, $D_S = 30$, $I = 300$, and $E = 5000$. Figure 5 presents the result of this experiment. We can observe that the combined UA-MCTS converges to a result much better than the Corrupted Model MCTS and close to the True Model MCTS. However, when trying to learn the transition function using the same network capacity, the agents could not find a path to the goal at all. The agent improves as the capacity of the transition function increases, but is still not able to outperform UA-MCTS with 16 units in its hidden layer.

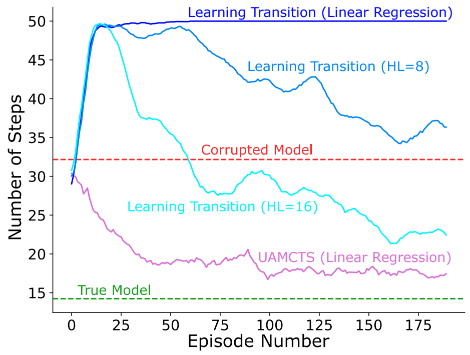

Figure 5: Comparison between learning the transition function with different hidden layer (HL) sizes and UA-MCTS method. The average number of steps to the goal is over 30 runs. The exploration parameter c for "True Model" and "Corrupted Model' is 2 and $\sqrt{2}$ respectively. For the other scenarios c is $\sqrt{2}$.

6 Related Work

There are numerous techniques which quantify and use uncertainty for MBRL. (Lütjens, Everett, and How 2019) captured uncertainty using ensembles of LSTM and Monte Carlo dropout, and changed the behaviour of their agent to act more cautiously in the uncertain parts by introducing a cost function for model predictive control (MPC), which can be considered a search algorithm.

Several approaches used uncertainty, but not as a component of an explicit search. (Abbas et al. 2020; Xiao et al. 2019; Buckman et al. 2018) modified the model value expansion (MVE) algorithm by taking model uncertainty into consideration. (Abbas et al. 2020) and (Xiao et al. 2019) selected the bootstrapping depth using model uncertainty. (Jafferjee et al. 2020) investigated the effect of model updates in both forward and backward directions with an imperfect model. (Lai et al. 2020) used forward and backward models in model-based policy optimization (Janner et al. 2019) in order to reduce accumulative model error while maintaining a similar update depth. (Talvitie 2017) designed a way to learn the model which reduces accumulative model error in deep lookahead.

CMAX and CMAX++ (Vemula et al. 2020; Vemula, Bagnell, and Likhachev 2021) are robust online path-planning algorithms in a real-time A* search framework, which can work around imperfect model predictions. As in our work, imperfect states are identified by comparing against the real environment during interactions. At runtime, CMAX completely disables parts of the model that are found to be in conflict. Examples of this include malfunctioning motors or overloaded robotic arms. To find a solution, at least one perfectly modelled path to the goal needs to exist. In contrast, UA-MCTS does not hard-prune such states, but shapes the search to prefer states with lower estimated uncertainty.

7 Conclusion and Future Work

UA-MCTS improves MCTS by learning a simple uncertainty model, instead of trying to learn corrections to a complex transition function. The method modifies the four main components of MCTS in a "risk-averse" way so that the search is directed away from the more uncertain parts of the model. To test UA-MCTS in practice, we developed a definition of uncertainty for a fixed given model, and a learned approximation that works with UA-MCTS. We tested UA-MCTS on the MinAtar testbed. Our results show that: 1) baseline MCTS suffers from severe performance degradation in the face of model uncertainty. 2) UA-MCTS outperforms MCTS for imperfect models. It can recover from performance degradation, or at least lessen its effects when used in conjunction with a learned uncertainty estimate. 3) The precision of the learned uncertainty model used by UA-MCTS has a very strong effect on agent performance. Improving this estimate requires further investigation. Other future directions include: investigating the impact of varying degrees of model imperfection on both UA-MCTS and baseline MCTS, especially the "risk-seeking" case where high uncertainty states must be explored, and comparing different uncertainty estimation techniques for UA-MCTS.

References

Abbas, Z.; Sokota, S.; Talvitie, E.; and White, M. 2020. Selective Dyna-style planning under limited model capacity. In *International Conference on Machine Learning*, 1–10. PMLR.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1): 1–43.

Buckman, J.; Hafner, D.; Tucker, G.; Brevdo, E.; and Lee, H. 2018. Sample-Efficient Reinforcement Learning with Stochastic Ensemble Value Expansion. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, 8234–8244. Curran Associates Inc.

Cazenave, T. 2010. Nested Monte-Carlo expression discovery. In *ECAI 2010*, 1057–1058. IOS Press.

Gao, C. 2020. *Search and Learning Algorithms for Two-Player Games with Application to the Game of Hex*. Ph.D. thesis, University of Alberta.

Gao, C.; Müller, M.; and Hayward, R. 2018. Three-Head Neural Network Architecture for Monte Carlo Tree Search. In *IJCAI*, 3762–3768.

Jafferjee, T.; Imani, E.; Talvitie, E.; White, M.; and Bowling, M. 2020. Hallucinating value: A pitfall of Dyna-style planning with imperfect environment models. *arXiv preprint arXiv:2006.04363*.

Janner, M.; Fu, J.; Zhang, M.; and Levine, S. 2019. When to Trust Your Model: Model-Based Policy Optimization. *Advances in Neural Information Processing Systems*, 32: 12519–12530.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kocsis, L.; Szepesvári, C.; and Willemson, J. 2006. Improved Monte-Carlo search. *Univ. Tartu, Estonia, Tech. Rep*, 1.

Kohankhaki, F.; Aghakasiri, K.; Zhang, H.; Wei, T.-H.; Gao, C.; and Müller, M. 2023. Monte Carlo Tree Search in the Presence of Transition Uncertainty. arXiv:2312.11348.

Lai, H.; Shen, J.; Zhang, W.; and Yu, Y. 2020. Bidirectional Model-based Policy Optimization. In *International Conference on Machine Learning*, 5618–5627. PMLR.

Lattimore, T.; and Szepesvári, C. 2020. *Bandit algorithms*. Cambridge University Press.

Lu, J.; Wang, X.; Wang, D.; and Wang, Y. 2016. Parallel Monte Carlo tree search in perfect information game with chance. In *2016 Chinese Control and Decision Conference (CCDC)*, 5050–5053. IEEE.

Lütjens, B.; Everett, M.; and How, J. P. 2019. Safe reinforcement learning with model uncertainty estimates. In *2019 International Conference on Robotics and Automation (ICRA)*, 8662–8668. IEEE.

Mansley, C.; Weinstein, A.; and Littman, M. 2011. Sample-based planning for continuous action Markov Decision Processes. In *Twenty-First International Conference on Automated Planning and Scheduling*.

Nijssen, J.; and Winands, M. 2010. Enhancements for multi-player Monte-Carlo tree search. In *International Conference on Computers and Games*, 238–249. Springer.

Sabharwal, A.; Samulowitz, H.; and Reddy, C. 2012. Guiding combinatorial optimization with UCT. In *International conference on integration of artificial intelligence (AI) and operations research (OR) techniques in constraint programming*, 356–361. Springer.

Saffidine, A. 2008. Utilisation dUCT au Hex. *Ecole Normale Super. Lyon, France, Tech. Rep*.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587): 484–489.

Silver, D.; Sutton, R. S.; and Müller, M. 2008. Sample-based learning and search with permanent and transient memories. In *Proceedings of the 25th International Conference on Machine Learning*, 968–975.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Talvitie, E. 2017. Self-correcting models for model-based Reinforcement Learning. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Vemula, A.; Bagnell, J. A.; and Likhachev, M. 2021. CMAX++: Leveraging Experience in Planning and Execution using Inaccurate Models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 6147–6155.

Vemula, A.; Oza, Y.; Bagnell, J. A.; and Likhachev, M. 2020. Planning and Execution using Inaccurate Models with Provable Guarantees. In *Proceedings of Robotics: Science and Systems*.

Xiao, C.; Wu, Y.; Ma, C.; Schuurmans, D.; and Müller, M. 2019. Learning to combat compounding-error in model-based reinforcement learning. *NeurIPS 2019 Deep Reinforcement Learning Workshop. arXiv preprint arXiv:1912.11206*.

Young, K.; and Tian, T. 2019. MinAtar: An Atari-inspired testbed for thorough and reproducible Reinforcement Learning experiments. *arXiv preprint arXiv:1903.03176*.